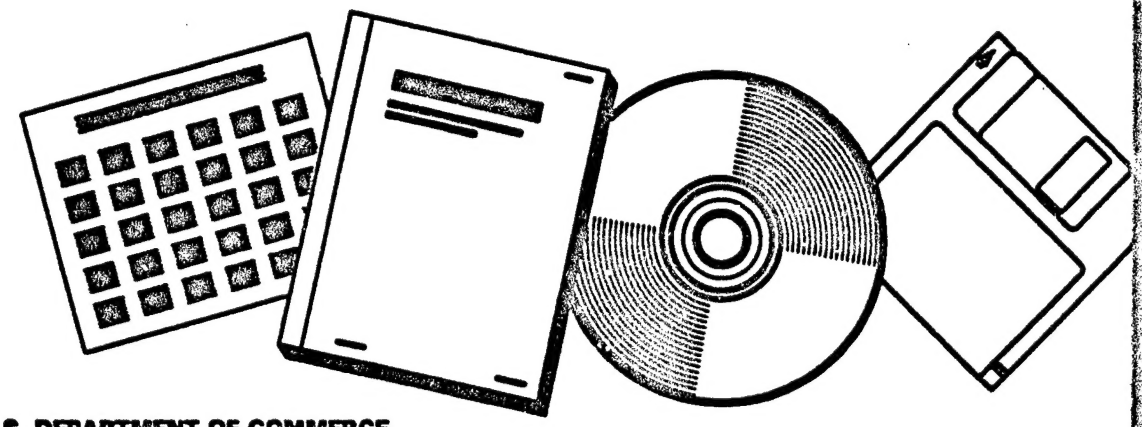
PB96-149893

SEMIGROUPS AND TRANSITIVE CLOSURE IN DEDUCTIVE DATABASES

STANFORD UNIV., CA

AUG 90



U.S. DEPARTMENT OF COMMERCE
National Technical Information Service

BIBLIOGRAPHIC INFORMATION

PB96-149893

Report Nos: STAN-CS-90-1327

Title: Semigroups and Transitive Closure in Deductive Databases.

Date: cAug 90

Authors: T. E. Plambeck.

Performing Organization: Stanford Univ., CA. Dept. of Computer Science.

Sponsoring Organization: *National Science Foundation, Washington, DC.*Air Force Office of Scientific Research, Bolling AFB, DC.

Contract Nos: NSF-IRI-87-22886, AFOSR-88-0266

Type of Report and Period Covered: Doctoral thesis.

NTIS Field/Group Codes: 62B (Computer Software)

Price: PC A07/MF A02

Availability: Available from the National Technical Information Service, Springfield, VA. 22161

Number of Pages: 133p

Keywords: *Data base management, *Semigroup theory, Recursive functions, Inference, Computation, Complexity, Factorization, Decomposition, Theses, *Deductive data bases, *Transitive closure.

Abstract: The thesis examines the transitive closure operation and more general linear recursive operations in deductive databases from a semigroup standpoint. An algebraic theory capable of completely characterizing all redundance encountered upon the expansion of linear recursive inference rules is first developed, and then the scope and computational complexity of the theory is studied. In addition, the authors sharpen and extend earlier results on efficient boundedness testing and more general query containment problems.

August 1990

Report No. STAN-CS-90-1327

Thesis

PB96-149893

SEMIGROUPS AND TRANSITIVE CLOSURE IN DEDUCTIVE DATABASES

by

Thane E. Plambeck

Department of Computer Science

Stanford University

Stanford, California 94305

SEMIGROUPS AND TRANSITIVE CLOSURE IN DEDUCTIVE DATABASES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Thane E. Plambeck
August 1990

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Jeffrey D. Ullman
(Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Moshe Y. Vardi

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

John C. Mitchell

Approved for the University Committee on Graduate Studies:

Dean of Graduate Studies

Abstract

This thesis examines the transitive closure operation and more general linear recursive operations in deductive databases from a semigroup standpoint. An algebraic theory capable of completely characterizing all redundance encountered upon the expansion of linear recursive inference rules is first developed, and then the scope and computational complexity of the theory is studied. In addition, we sharpen and extend earlier results on efficient boundedness testing and more general query containment problems.

Acknowledgements

I would first like to thank my principal dissertation advisor, Jeff Ullman, for telling me that I would get to tell him my thesis topic, and not conversely. He introduced me to the problems considered in this thesis and was always readily available for discussion and guidance. I also thank the other members of my thesis committee, John Mitchell, Nils Nilsson, and particularly Moshe Vardi for help in shaping the contents of this thesis.

Anil Gangolli, Yehoshua Sagiv, Ashok Subramanian, and especially Yatin Saraiya I would also like to thank for pointing out my errors and for their remarkable willingness to be repeatedly plunged into the details of my work.

My years at Stanford have been altogether congenial ones, and I have made many new friends. As C. G. Plaxton has suggested, let's hope we cross paths again.

Finally, I thank my parents, Vernon Lewis and Marlene Loyola Plambeck, who got the ball rolling, and for whom "Eng." will always refer to "English," and not "Engineering."

This work has been supported by NSF grant IRI-87-22886, AFOSR grant 88-0266, and a grant of IBM.

And thank you, Gloria!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis is an examination of the interplay between three ideas: relational databases, transitive closure operations, and semigroups.

1.1 Motivation

Recently, there has been a growing realization that large scale relational database systems will achieve their maximum flexibility and usefulness only if they allow sophisticated querying operations. Because logic-based methods can be used not only to specify the simplest querying operations, but also can be employed as full-fledged programming languages in their own right, much attention has been given to the possibility of extending the more primitive relational querying operations (e.g., select, project, join) to include more advanced constructs from logic programming languages, such as Prolog [Cloc81]. In particular, because first-order relational database query languages lack expressive power [AhUl79], the use of *recursion* as a query primitive has received increasing attention in the database community [ChHa85], [GMN84], [Ullm85]. Unfortunately, the addition of recursion to a query language carries a heavy penalty when we come to query optimization, because it has been discovered [Vard88], [GMSV87] that many natural questions one would like answered about general recursive programs, even in the more simple querying contexts, are either combinatorially difficult [SaYa80] or even undecidable [Vard88]. Recent authors have therefore turned to the identification of restricted classes of recursive programs for which particular optimizations can be performed efficiently [Ioan89], [Sar89a], [RSUV89], and much useful effort has been made to isolate what it is that makes a given program difficult to optimize.

Given a database query, how does one compute the answer in the most efficient way? Unfortunately, even the first step of defining what we might mean by "the most efficient way" is problematic [Ban86a]. Should one attempt to define the complexity of a query in terms of the query itself, or should the underlying database play a role in the definition as well? Because one may construct examples to show how the varying structure of the underlying database can exert a heavy influence not only on the complexity of alternative evaluation algorithms, but also on the size of the resulting query answers, one is tempted to include a consideration of the size and structure of the database in the definition of query complexity, and indeed this approach has its advantages [ChHa82]. However, there is a parallel investigative line that has also been taken. We may simply describe it as *syntactic query analysis*. Here, authors have attempted to divorce the analysis of a query from its underlying database, and they have shown how strong results about the complexity of evaluating a recursive query may be won by considering its *recursive expansions* in conjunction with the idea of *conjunctive query containment*. In their strongest possible forms, syntactic analysis results may imply either that a recursively defined query is in fact replaceable by a nonrecursively defined one—a *boundedness* result—or they may make clear some aspect of the *redundancy* inherent in recursive expansion of queries that is relevant to the efficiency of most recursive query evaluation algorithms [Banc86b].

Here, our focus will be restricted to the simplest manifestations of recursive queries, the *linear recursive* queries. For us, such queries will manifest themselves as pure *Horn clause* rules without function symbols or negation; their syntax will conform to the *Datalog* language, which is a syntactic subset of Prolog more fully described below. At the core of the syntactic analysis of such queries is the question: given a set of linear recursive queries, how can we recognize the query containments between their various recursive expansions? Unfortunately, the general problem of recognizing query containment is NP-complete [ChMe77], so a retreat from the search for efficient and all-encompassing theories is well advised. Moreover, even some of the simplest possible *restricted* questions about query containments between general recursive query expansions have been remarkably resistant to the attacks of several authors, and one need not look far to find undecidability results for particular problems.

The present thesis adds to the growing body of work on syntactic query analysis by introducing a set of constructive techniques for query containment problems that we may loosely describe as *semigroup methods*. At the center is the observation that the recursive

expansion of linear recursive rules can be viewed as an abstract *associative* operation whose inverse or *factorization* properties may reveal the structure of query containments. Our methods will have the advantage that when they work completely, they will give us a complete and succinct description of all the query containments that are encountered upon recursive expansion of a set of queries; the query containment information will come to us in the form of a *semigroup presentation*. Such information will impinge not only on the often-studied boundedness properties of queries, but also on *rule commutativity* and *recursive redundance* properties that have been suggested by earlier authors in syntactic query analysis contexts. Our results will lead us both to new syntactic analysis problems as well as to reconsiderations of previously proposed issues, and we shall also have some new things to say about some of the simplest manifestations of linear recursive queries.

1.2 Databases and Datalog

We shall take Codd's relational model [Codd70] of databases as our starting point. The following basic definitions follow those to be found for example in [Sagi87].

We shall think of the information stored in a database as represented as a finite number of finite mathematical relations. More formally, a relation Q for a predicate q is a set of *ground atoms* of some fixed arity. Such atoms have only constants and no variables occurring in them—for example, $q(4, john, 2, 2)$ might be a typical ground atom. A collection of such relations (which may have different arities) is then called a *database*. If $Q_1, Q_2, \ldots, Q_l$ are relations for predicates $q_1, q_2, \ldots, q_l$, then the set of all the ground atoms that occur in the various relations Q_i is called an *interpretation* or a *structure*.

A *Datalog program* is a finite set of *Horn clause rules* (see, for example, [Ullm88]), each of which is required to have only predicates, variables, and constants in its head and body. Function symbols, negation, as well as other common constructs from logic programming languages (for example, lists, Prolog cuts, or arithmetic operations) are not allowed. The Horn-clause rules defining a Datalog program are often more simply referred to as *rules*. Each rule has a *head*, appearing on the left-hand side of the symbol $:-$, and a *body*, appearing on the right-hand side of the $:-$ symbol. The head of a rule is required to be a single *atomic formula* or simply *atom*, that is, a *predicate* with *variables* in each of its argument positions. The body of a rule is a (possibly empty) conjunction of atoms. A rule with an empty body is also sometimes called a *fact*.

Suppose Π is a fixed Datalog program. If a predicate p occurs as the head predicate of some rule of Π, then p is called an *intensional database predicate*, (or IDB predicate). Predicates q of Π not occurring in any rule head are called *extensional database predicates*, (or EDB predicates). The *input* to a Datalog program Π is a fixed relation Q for each extensional predicate q of Π, and the collection of all such relations is called the *extensional database*. The *output* computed by Π is in effect a relation for each intensional predicate, and it is called the *intensional database*.

Example 1.1 The following Datalog program Π has two rules, r_1 and r_2:

r_1 : ancestor(XY) :- father(XY)

r_2 : ancestor(XY) :- ancestor(XA) & father(AY)

Here, the predicate father is extensional, while the predicate ancestor is intensional. If we additionally specify the particular input extensional database relation

father(Paul, Paul Jr.)

father(Paul Jr., Paul III),

then the output computed by Π is the relation

ancestor(Paul, Paul Jr.)

ancestor(Paul Jr., Paul III)

ancestor(Paul, Paul III).

■

The intuition that the output of Datalog program should be definable from its input by starting with its input EDB relations and then applying rules exhaustively to derive all output IDB relations can be made mathematically precise, and in fact it shall be our assumption throughout that the method of *fixed point evaluation* [Ullm88] is used to assign a unique *minimal model* to the relations of a Datalog program, given its input. It is known that the minimal model containing any given EDB relations is unique, and that it corresponds exactly to the set of facts one can derive, using the rules, from the given database. We again refer the reader to [Ullm88] for a more detailed explanation of the minimal model construction.

1.3 Linear recursion

If the head predicate p of a Datalog rule r also appears in its body, then the rule r is said to be *recursive*. If r is recursive with head predicate p, and p occurs only once in the body of r, then we shall say that r is a rule *linear recursive in* p. Regardless of whether r is recursive or not, the variables of r that occur in the head of the rule are called its *distinguished variables*; other variables that occur only in the body of r are called *nondistinguished variables*. Except where otherwise noted, we shall always be considering Datalog programs consisting of one or more linear recursive rules, with each linear recursive in a single predicate p.

1.4 Semigroups

A *semigroup* $(S, \cdot)$ is a set S with a binary operation, denoted by $\cdot$, satisfying the *associativity law*: $x \cdot (y \cdot z) = (x \cdot y) \cdot z$. If S additionally has an identity element 1 satisfying $x \cdot 1 = 1 \cdot x = x$ for all elements $x \in S$, then S is called a *monoid*.

In this thesis we shall be principally concerned with semigroups and monoids that arise in the analysis of transitive closure-like rules. Among the simplest of such semigroups are the *free semigroups*.

1.4.1 Free semigroups

Let A be a set called an *alphabet*, the elements of A to be called *letters*. A *word* in A is a nonempty finite sequence $x_1x_2\cdots x_n$ of elements of A. Two words $x_1x_2\cdots x_n$ and $y_1y_2\cdots y_m$ are defined to be equal if and only if they coincide as sequences—that is, we must have $m = n$ and $x_i = y_i$ for $1 \leq i \leq n$. We shall use the notation A^+ to denote the set of all words in the alphabet A. If we define a binary operation $\cdot$ on A^+ by the *concatenation* operation

$$(x_1x_2\cdots x_n)(y_1y_2\cdots y_m) = x_1x_2\cdots x_ny_1y_2\cdots y_m,$$

then $(A^+, \cdot)$ is a semigroup which we shall call the *free semigroup* on A. The implication

$$x_1x_2\cdots x_n = y_1y_2\cdots y_m \Rightarrow m = n \text{ and } x_i = y_i \text{ for } 1 \leq i \leq n$$

expresses the fact that every word $w \in A^+$ has a *unique factorization* as a product of elements of A.

A *subsemigroup* T of a semigroup $(S, \cdot)$ is a subset of S that is closed under $\cdot$, that is, $x_1, x_2 \in T$ implies $x_1 x_2 \in T$. If A is a subset of a semigroup S, then there is a smallest subsemigroup of S containing A, which we shall call the *subsemigroup generated by* A, and we shall use the notation $\langle A \rangle$ to stand for this semigroup. The semigroup $\langle A \rangle$ coincides with the set of all finite products of elements of A. Any subset A of S such that $\langle A \rangle = S$ is called a set of *generators* of S; there always exists such a set, for example, we may take $A = S$. If S is a free semigroup on the alphabet A, then $\langle A \rangle = A^+$ and A is a set of generators of S.

A *homomorphism* ψ from a semigroup $(S, \cdot)$ to a semigroup $(T, \circ)$ is a mapping ψ from S into T such that $\psi(x \cdot y) = \psi(x) \circ \psi(y)$ for all $x, y \in S$. The following elementary result [Lall79] connects the ideas of free and general semigroups via the use of homomorphisms:

Theorem 1.2 *If ψ is a fixed mapping from a set A into a semigroup S, then there exists a unique homomorphism $\dot{\psi}$ from the free semigroup A^+ into S such that $\dot{\psi}(x) = \psi(x)$ for all $x \in A$. Also, $\dot{\psi}$ is onto if and only if $\psi(A)$ is a set of generators of S.*

Proof For $w = x_1 x_2 \cdots x_n \in A^+$, define $\dot{\psi}(w) = \psi(x_1)\psi(x_2)\cdots\psi(x_n)$. The mapping $\dot{\psi}$ is therefore forced to be a homomorphism, and we see in fact that if $\dot{\psi}$ is to be a homomorphism at all, then the given definition is forced on us. So $\dot{\psi}$ is unique. Finally, $\psi(A)$ will be a set of generators of S if and only if each $s = \psi(y_1)\psi(y_2)\cdots\psi(y_m) = \dot{\psi}(y_1 y_2 \cdots y_m)$ for some $y_1, \cdots, y_m \in A$. Equivalently, $\dot{\psi}$ must be onto. ∎

1.4.2 Presentations

Theorem 1.2, when applied to a set A of generators of a semigroup S, yields the following corollary.

Corollary 1.3 *Every semigroup S is a homomorphic image of the free semigroup A^+ on any set A of generators of S.*

The corollary is important because it lies at the root of the idea of a *semigroup presentation*, which it is our next task to describe.

The idea of a semigroup presentation arose most importantly in the context of certain decidability problems in logic [Thue12], [Post47]. Our principal interest in presentations will be to use them to capture redundancy occurring amongst the recursive expansions of a

database query—but here we anticipate ourselves a bit. For now, all we need to do is pick up the basic definitions.

Let S be a semigroup. A set A is called a set of *generating symbols for S under the mapping $\psi : A \to S$* if the homomorphic extension $\hat{\psi}$ to the free semigroup A^+ (given by defining $\hat{\psi}(x_1 \cdots x_n) = \psi(x_1) \cdots \psi(x_n)$ as in Theorem 1.2) is onto. If A is a set of generating symbols for S under ψ, then when two words $w_1, w_2 \in A^+$ satisfy $\hat{\psi}(w_1) = \hat{\psi}(w_2)$, we say that S satisfies the *relation* $w_1 = w_2$, or that $w_1 = w_2$ is a relation in S.

If we are given a set of relations $\{w_i = w_i',\ i \in I\}$ in S, then we define a word $v \in A^+$ to be *directly derivable* from a second word $v' \in A^+$ if either $v = rw_is$ and $v' = rw_i's$, or $v = rw_i's$ and $v' = rw_is$ for some $i \in I$ and $r, s \in A^+$. When $v = v_0, v_1, \cdots, v_m = v'$ is a sequence of words, each member of the sequence directly derivable from its predecessor and successor (if they exist), then we say that v is *derivable* from v' (in n steps).

If v is derivable from v' under a set of relations $\{w_i = w_i',\ i \in I\}$ in S, then $v = v'$ is itself a relation in S. We shall say therefore that $v = v'$ is *consequence* of the relations $\{w_i = w_i',\ i \in I\}$ in S, or sometimes that $v = v'$ is *implied by* the given relations. If *all* relations in S are consequences of $\{w_i = w_i',\ i \in I\}$, then $\langle A \mid \{w_i = w_i',\ i \in I\}\rangle$ is called a *presentation of S defined by ψ*.

Thus, to give an example of a semigroup S it is enough to give a set of *generators* A and *relations* $\{w_i = w_i',\ i \in I\}$. To be absolutely precise, we should say that the semigroup S is in fact defined to be the *quotient* of A^+ by the *congruence on* A^+ generated by all pairs (w_i, w_i'), with $i \in I$. Here, a *congruence* is simply an equivalence relation ρ on S that is stable under left and right multiplication, and the elements of a *quotient* semigroup are defined to be the equivalence classes of a congruence with multiplication between elements inherited from S. We shall not concern ourselves with the details of this *congruence-quotient construction* here because we shall carry it out in detail for our particular *rule expansion semigroups*, in Chapter 2, below; for precise definitions of these terms and a verification congruence-quotient construction in the general case, consult [Lall79].

1.4.3 Transformation semigroups

In addition to semigroups presented by generators and relations, a second type of semigroup will arise frequently in our work. These are the *transformation semigroups*. Here we are again just interested in the basic definitions.

Suppose that n is a positive integer, and let $I_n = \{1, 2, \ldots, n\}$. A *transformation* of I_n

is just a function $f : I_n \to I_n$. If f is onto, then f is called a *permutation*. The set of all transformations of I_n admits an associative multiplication defined by function composition, and if F is a set of transformations of I_n closed under function composition, then we call F a *transformation semigroup*.

One natural way to represent a given transformation f is to use a directed *substitution graph* on the vertices I_n, with a directed edge $i \to j$ if $f(i) = j$. We see immediately that the substitution graph of f has uniform outdegree 1, and that if f is a permutation, then the substitution graph of f partitions into directed cycles, some of which may be trivial directed cycles (i.e., loop edges) corresponding to fixed points of f. The substitution graph of a general transformation f may have vertices belonging to no directed cycle, although if i is such a vertex, then there is a minimal $l \geq 1$ such that $f^l(i)$ is contained in some directed cycle of the substitution graph. In this case, the vertex i is called *stem*, and l is called its *height*. Vertices belonging to directed cycles of the substitution graph are defined to have height zero.

For example, let f be the transformation of I_{10} defined by the mapping

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 4 & 2 & 1 & 7 & 3 & 10 & 1 & 5 & 7 & 10 \end{pmatrix}.$$

The substitution graph of f has three directed cycles. Two of these are trivial directed cycles involving the fixed points 2 and 10; the third is a directed cycle $1 \to 4 \to 7 \to 1$ on the three vertices 1, 4, and 7 in this order. The remaining vertices 3, 5, 6, 8, and 9 are stem. Of these, 3, 6, and 9 have height 1; the vertex 5 has height 2, while 8 has height 3.

1.5 Overview

In this section we survey our main results and describe the thesis organization. Some technical terms are necessarily left undefined in the present overview; the most important terminology appears in italics. We shall also relate some earlier work to our own.

1.5.1 Thesis organization and main results

Chapter 2 introduces the basic mathematical object studied in this thesis, the *rule expansion semigroup*. Roughly, the elements of a rule expansion semigroup consist of the set of all top-down *recursive expansions* of a *rule set* S of linear recursive Datalog rules, every such

rule being linear recursive in a single IDB relation p. The *multiplication* in a rule expansion semigroup is defined by *rule expansion*. In Section 2.2.3 we introduce a simple method for the calculation of rule expansions in the presence of multiple recursive rules, and call the method *syntactic expansion*.

There is a natural *partial ordering* $\succeq$ *by query containment* on the elements of rule expansion semigroups; most problems that arise in syntactic query analysis can be phrased in terms of questions about this partial ordering. An important elementary property of the query containment ordering that is exploited throughout the thesis is that *recursive expansion respects query containment*; a proof appears in our "Splicing Lemma," Lemma 2.4. We close our introductory chapter with a few simple examples of rule expansion semigroups, and give a brief discussion of how *basis rules* fit into our scheme.

Detecting query containments amongst the various recursive expansions of a rule set is in general a difficult problem; for example, a recent result of Saraiya [Sar89b] shows that deciding even the simplest *commutative containment relationship* $r_1 r_2 \succeq r_2 r_1$ is NP-complete in the general case. However, in Chapter 3 we show how it is possible under certain circumstances to win a complete description of all query containments encountered upon recursive expansion of a set of rules by appealing to the ideas of *free rules* and *rule factorization*. Basically, the idea is that a rule is free if it satisfies *no* query containments, and if the elements of a given rule set can be written as recursive expansions of free rules, then methods from free semigroup theory may be used to give a complete description of all query containments in terms of a *semigroup presentation*. Such information often reveals information about the *rule commutativity* or *boundedness* properties of recursive queries, and we introduce these issues.

In Chapter 4 we begin a study of the applicability of our techniques in the general case by focusing on the query containment structure of a single linear recursive rule. There is a close connection between the ideas of rule freeness and *sirup uniform boundedness* here; in particular, a recent difficult decidability result of Vardi [Vard90] suggests that the true complexity of our techniques may be "just on the side of decidability." Our own contribution here is first a technical *containment depth* result that gives us some limited information about the query containment structure of a single rule; more importantly, the notation we introduce will allow us to later present a single efficiently testable sufficient condition for sirup unboundedness that simultaneously subsumes several earlier criteria for this problem. There is a close connection between the *transformation structure* of the *substitution graph* of

a rule and the EDB subgoals of its recursive expansions that can be exploited in analyzing boundedness; in particular, in our *cut* and *cut completion* conditions, *maximal height stem variables* play a critical role.

In Chapter 5 we turn to the other side applicability issues when we take up the problem of *rule factorization*. Intuitively, a linear rule r factors if it can be obtained as a recursive expansion of simpler rules r_1 and r_2. Because our techniques from Chapter 3 assume such factorizations to be given, the factorization problem is basic to the applicability of our techniques. We begin by introducing the idea of an *irreducible* rule, and we show that every rule has a factorization into irreducibles. Assuming first that the arity of a linear recursive rule r is bounded, we give an algorithm to factor r in polynomial time; we shall call our method the Γ-*multigraph algorithm*. When the arity of r is not bounded, the Γ-multigraph method fails to run in time polynomial in the size of the rule, so we develop a second method called the *flow graph algorithm* to factor general rules in polynomial time as well, albeit at a sometimes worse time complexity than that in the bounded arity case. Next, we consider *unique factorization*, and we show that although unique factorization properties are usually not present in rule expansion semigroups, nevertheless there are particular situations when a weaker notion of unique factorizations exists, and we show how to exploit these factorizations by using the idea *graph semigroups*, which reappear in Chapter 6 as well.

Chapter 6 demonstrates how semigroup techniques may be brought to bear on the study of the query containments of *linear chain rules*, which are a simple class of rules that arise in both practical and theoretical settings. The thesis closes with some open questions and speculation about as yet unproven conjectures.

1.5.2 Related work

The work most similar to our own is the Ph.D. thesis of Jeff Naughton [Naug87]. Most of our elementary definitions have been taken directly either from his thesis, or alternatively from a sequence of papers Naughton wrote in conjunction with Yehoshua Sagiv [Naug86b], [Naug86a], [NaSa87], [NaSa88], [Naug88]. Most importantly, the concepts of uniform boundedness and boundedness, and the idea of a *chain* in the context of recursive expansions of linear recursive rules (arising for us in Chapter 4) can be traced to Naughton and Sagiv. Even the idea of a free rule set, which is central to the present work, has its genesis in an observation of Naughton that containment freedom between recursive expansions is "a useful property." [Naug87]. Here, we have taken Naughton's comment, and have expanded

it into another thesis!

Other work closely related to our own can be found in the papers of Yannis Ioannidis [Ioan85], [Ioan89], [IoRa88]. In particular, interesting discussions of rule commutativity and its consequences for the processing of linear recursion can be found in [Ioan89], and also in [RSUV89]. Ioannidis and Wong [IoWo88] give an algebraic account of linear recursion in relational databases that in part anticipates our own rule expansion semigroup definition in Chapter 2, below. With a few exceptions, all the semigroup theory we employ can be found in the clear exposition of Lallement [Lall79].

A recent Ph.D. thesis by Guozhu Dong [Dong88] uses language quite similar to our own. Dong writes that his object is to study "the composition of Datalog mappings in order to analyze the situation where a sequence of Datalog program queries are evaluated serially," and then he goes on to study the "reverse process of composition," which he calls *decomposition*. He develops the notion of a *prime* program as one which cannot be decomposed, and then proves results relating boundedness to the decomposability of programs into single rule primes. In its general spirit and outline, Dong's work bears a resemblance to our own, in particular because commutativity and unique factorization properties arise naturally for him, as they will for us. The difference between Dong's work and our own, however, stems from first principles—while our notion of "composition" corresponds to recursive expansion of rules, Dong's composition is relative to *cascading execution* of complete Datalog programs. For example, Dong would say that a single program Π decomposes into single rule programs Π_1 and Π_2 if the relation computed by Π is obtainable by computing the fixed point of Π_1 and then using this fixpoint as a starting point for a second Π_2 fixed point computation, while we would say that Π decomposes (actually *factors*, in our language) if Π is the rule obtained by recursively expanding the rule Π_1 by the rule Π_2. The two notions do not appear to overlap—for example, it is not even known whether testing for indecomposibility in Dong's sense is decidable, while our *rule factorization into irreducibles* is decidable, and in fact can be carried out in time polynomial in the size of the rule.

Still more discussion of other authors' work and its relation to our own will be found at various places throughout our work.

Chapter 2

Rule expansion semigroups

Our basic mathematical object of study is the *rule expansion semigroup*. Roughly, the elements of a rule expansion semigroup consist of the set $\mathcal{J}(S)$ of top-down expansions of a set S of linear recursive Datalog rules, with "multiplication" defined by rule expansion. A precise definition and some examples appear in Section 2.2.4 below.

Two points should be made. First, we shall consider only *finite* sets of rules S. The restriction is a natural one: real programs are finite. Second, to ensure that two elements of $\mathcal{J}(S)$ can always be "combined," (i.e., multiplied), we shall restrict our attention only to sets S containing *linear recursive* rules over a *single* recursive predicate p. Finally, we shall always assume the head predicate of such a recursive rule to consist of *distinct* variables. Even with our restrictions, we shall see not only that a rule expansion semigroup $\mathcal{J}(S)$ can be a mathematical object of considerable complexity—for example, every finite group is isomorphic to a rule expansion semigroup—but also that our methods will apply to the study of more simple and "natural" Datalog recursions, such as the simple transitive closure rule for graphs (see Section 2.1).

Corresponding to every rule expansion semigroup $\mathcal{J}(S)$ there is a natural partial ordering $\preceq$ given by query containment. We take up the details in Section 2.2.

Important to our viewpoint is the implicit restriction that the generating set S of a rule expansion semigroup contain no basis (that is, nonrecursive) rules. Our motivation for this simplification is twofold. First, by removing the basis rules from a program, we endow its recursive rules with a multiplicative structure that can be studied independently from its basis rules. Second, often basis rules "make no difference," in the sense that their reintroduction into a program often does not change the query containment properties that are

important in recursive query processing applications: for example, when all nonrecursive rule bodies contain only EDB subgoals that appear in the body of no recursive rule, then it can be shown (see, for example, [Ioan85]) that studying the query containments between expansions of the recursive rules only suffices to capture all query containments. In Section 2.3 we return for a moment to consider basis rules, and introduce questions which will be studied in more detail later.

2.1 Linear rules and top-down expansion

A natural starting point is the simple transitive closure rule for directed graphs:

$$u : \mathrm{p}(\mathtt{XY}) \ :- \ \mathrm{p}(\mathtt{XA}) \ \& \ \mathrm{e}(\mathtt{AY}). \tag{2.1}$$

If we additionally take the basis rule

$$b : \mathrm{p}(\mathtt{XY}) \ :- \ \mathrm{e}(\mathtt{XY}),$$

we can think of the program $\Pi = \{u, b\}$ as defining a "directed path" relation p from a given directed "arc" relation e.

Let $S = \{u\}$. Expanding the rule u by itself recursively, we obtain an infinite set $\mathcal{J}(S)$ of rules

$$\begin{aligned} u &: \mathrm{p}(\mathtt{XY}) \ :- \ \mathrm{p}(\mathtt{XA}) \ \& \ \mathrm{e}(\mathtt{AY}), \\ u^2 &: \mathrm{p}(\mathtt{XY}) \ :- \ \mathrm{p}(\mathtt{XA'}) \ \& \ \mathrm{e}(\mathtt{A'A}) \ \& \ \mathrm{e}(\mathtt{AY}), \\ u^3 &: \mathrm{p}(\mathtt{XY}) \ :- \ \mathrm{p}(\mathtt{XA''}) \ \& \ \mathrm{e}(\mathtt{A''A'}) \ \& \ \mathrm{e}(\mathtt{A'A}) \ \& \ \mathrm{e}(\mathtt{AY}), \end{aligned}$$

together with the general rule for every $i \geq 3$,

$$u^i : \mathrm{p}(\mathtt{XY}) \ :- \ \mathrm{p}(\mathtt{XA}^{(i-1)}) \ \& \ \mathrm{e}(\mathtt{A}^{(i-1)}\mathtt{A}^{(i-2)}) \ \& \ \cdots \ \& \ \mathrm{e}(\mathtt{A'A}) \ \& \ \mathrm{e}(\mathtt{AY}). \tag{2.2}$$

The names $u, u^2, u^3, \ldots$ we have given to these rules are meant to be suggestive: in some sense there is a correspondence between this set $\mathcal{J}(S)$ of top-down rule expansions and the free semigroup $\{u, uu, uuu, \ldots\}$ generated by the formal symbol u (with "multiplication" corresponding to string concatenation). In Example 2.7 below, we make this correspondence precise.

2.2 Partial ordering by query containment

To continue, we need a slightly more complicated example. Let $S = \{s, t\}$ consist of the two rules

$$s : \mathtt{p(XY)} \ :- \ \mathtt{p(BA)} \ \& \ \mathtt{e(XB)} \ \& \ \mathtt{e(AY)} \tag{2.3}$$

$$t : \mathtt{p(XY)} \ :- \ \mathtt{p(AB)} \ \& \ \mathtt{e(BY)} \ \& \ \mathtt{e(YA)} \ \& \ \mathtt{e(XA)}, \tag{2.4}$$

and suppose that $\mathcal{P}$ and $\mathcal{E}$ are fixed finite sets containing initial sets of p- and e-facts, respectively. Then we can use the two rules s and t to compute new p-facts. For example, let

$$\mathcal{P} \ = \ \{p(2,5), p(2,4)\}$$

be an initial set of p-facts, and let

$$\mathcal{E} \ = \ \{e(4,1), e(1,2), e(3,2), e(5,3), e(1,6), e(2,2)\}$$

be an initial set of e-facts. Then by using the rule t and the facts $p(2,5)$, $e(5,3)$, $e(3,2)$, and $e(2,2)$, we can infer the p-fact $p(2,3)$:

$$t : \mathtt{p(2,3)} \ :- \ \mathtt{p(2,5)} \ \& \ \mathtt{e(5,3)} \ \& \ \mathtt{e(3,2)} \ \& \ \mathtt{e(2,2)}. \tag{2.5}$$

Here, we have used the bindings $\mathtt{X} = 2$, $\mathtt{Y} = 3$, $\mathtt{A} = 2$, and $\mathtt{B} = 5$.

More facts can be proved by recursive expansion of the rules s and t. For example, substituting the rule t for the p-subgoal in the rule s, we obtain the rule

$$st : \mathtt{p(XY)} \ :- \ \mathtt{p(A'B')} \ \& \ \mathtt{e(B'A)} \ \& \ \mathtt{e(AA')} \ \& \ \mathtt{e(BA')} \ \& \ \mathtt{e(XB)} \ \& \ \mathtt{e(AY)}. \tag{2.6}$$

We have given the name "st" to this rule in order to suggest that it is obtained by "multiplying" (i.e. expanding) the rule s by the rule t. We can use the rule st to infer the p-fact $p(5,6)$: we have

$$st : \mathtt{p(5,6)} \ :- \ \mathtt{p(2,4)} \ \& \ \mathtt{e(4,1)} \ \& \ \mathtt{e(1,2)} \ \& \ \mathtt{e(3,2)} \ \& \ \mathtt{e(5,3)} \ \& \ \mathtt{e(1,6)},$$

where we have used the bindings $\mathtt{A} = 1$, $\mathtt{A}' = 2$, $\mathtt{B} = 3$, $\mathtt{B}' = 4$, $\mathtt{X} = 5$, and $\mathtt{Y} = 6$. Naively, we may expect that given arbitrary initial fact sets $\mathcal{P}$ and $\mathcal{E}$, we may be forced to expand the two rules s and t in arbitrary combinations in order to find all the p-facts implied by these two rules. However, in this particular case, a considerable simplification is possible. In fact, any p-fact which can be inferred using the rules s and t together can be inferred by using the rule s alone. To see why, we need to develop the idea of a *containment mapping*.

2.2.1 Containment mappings

We return to the rules s and t above (equations 2.3 and 2.4). Let ϕ be the function mapping the set of head and body predicates of s into the set of head and body predicates of t as follows:

$$\begin{aligned}
\phi(\mathtt{p(XY)}) &= \mathtt{p(XY)} \\
\phi(\mathtt{e(BA)}) &= \mathtt{e(AB)} \\
\phi(\mathtt{e(XB)}) &= \mathtt{e(XA)} \\
\phi(\mathtt{e(AY)}) &= \mathtt{e(BY)}.
\end{aligned}$$

The mapping ϕ satisfies the following four properties:

1. ϕ maps the head predicate of s to the head predicate of t, and these predicates have the same name.

2. ϕ maps each body predicate of s to a body predicate of t with the same name.

3. If q_1 and q_2 are predicates of s with identical distinguished variables at positions i and j, respectively, then $\phi(q_1)$ and $\phi(q_2)$ have identical distinguished variables at positions i and j, respectively.

4. If q_1 and q_2 are predicates of s with identical nondistinguished variables at positions i and j, respectively, then $\phi(q_1)$ and $\phi(q_2)$ have identical variables (distinguished or nondistinguished) at positions i and j, respectively.

A mapping like ϕ, satisfying conditions 1-4 above, is called a *containment mapping.*

Corresponding to every containment mapping ϕ from a rule r_1 to a rule r_2 there is a unique *derived mapping* $\dot{\phi}$ which maps the set of variables of r_1 into the set of variables of r_2, and agrees with ϕ at the predicate variable level. For the particular containment mapping ϕ above, the derived mapping $\dot{\phi}$ is given by

$$\begin{aligned}
\dot{\phi}(\mathtt{X}) &= \mathtt{X} \\
\dot{\phi}(\mathtt{Y}) &= \mathtt{Y} \\
\dot{\phi}(\mathtt{A}) &= \mathtt{B} \\
\dot{\phi}(\mathtt{B}) &= \mathtt{A}.
\end{aligned}$$

When a containment mapping $\phi : r_1 \rightarrow r_2$ from a rule r_1 to a rule r_2 exists, then any fact that can be concluded by using one application of the rule r_2 can also be concluded by using one application of the rule r_1. (The converse is also true: see Theorem 2.1 below). For example, consider the derivation 2.5 above, where it was shown how to obtain the p-fact $p(2,3)$ using the rule t. The containment mapping ϕ above shows us how to derive this same fact using the rule s. Here are the details. For each subgoal in the body of s, we use the derived mapping $\dot{\phi}$ and t-derivation of $p(2,3)$ to find an s-derivation of this fact. The t-derivation of the $p(2,3)$ was given in Equation 2.5, and had bindings $\mathtt{X} = 2$, $\mathtt{Y} = 3$, $\mathtt{A} = 2$, and $\mathtt{B} = 5$. The derived mapping $\dot{\phi}$ is given by

$$\dot{\phi}(\mathtt{X}) = \mathtt{X} = 2$$
$$\dot{\phi}(\mathtt{Y}) = \mathtt{Y} = 3$$
$$\dot{\phi}(\mathtt{A}) = \mathtt{B} = 5$$
$$\dot{\phi}(\mathtt{B}) = \mathtt{A} = 2$$

The associated s-derivation is then

$$s : \mathtt{p}(\dot{\phi}(\mathtt{X}), \dot{\phi}(\mathtt{Y})) \ :- \ \mathtt{p}(\dot{\phi}(\mathtt{B}), \dot{\phi}(\mathtt{A})) \ \& \ \mathtt{e}(\dot{\phi}(\mathtt{X}), \dot{\phi}(\mathtt{B})) \ \& \ \mathtt{e}(\dot{\phi}(\mathtt{A}), \dot{\phi}(\mathtt{Y})), \qquad (2.7)$$

or equivalently,

$$s : \mathtt{p}(2,3) \ :- \ \mathtt{p}(2,5) \ \& \ \mathtt{e}(2,2) \ \& \ \mathtt{e}(5,3).$$

To show the nonexistence of a containment mapping ϕ from a rule r_1 to a rule r_2, it is often convenient to work with derived mappings. For example, consider the two top-down expansions u^2 and u^3 of the simple transitive closure rule 2.1, above:

$$u^2 : \mathtt{p}(\mathtt{XY}) \ :- \ \mathtt{p}(\mathtt{XA}') \ \& \ \mathtt{e}(\mathtt{A}'\mathtt{A}) \ \& \ \mathtt{e}(\mathtt{AY}),$$
$$u^3 : \mathtt{p}(\mathtt{XY}) \ :- \ \mathtt{p}(\mathtt{XA}'') \ \& \ \mathtt{e}(\mathtt{A}''\mathtt{A}') \ \& \ \mathtt{e}(\mathtt{A}'\mathtt{A}) \ \& \ \mathtt{e}(\mathtt{AY}).$$

We claim there is no containment mapping "either way" between these two rules. Here, we shall show only that there is no containment mapping $\phi : u^2 \rightarrow u^3$ (see Example 3.3, below, for the converse in a generalized form).

Suppose there were a containment mapping $\phi : u^2 \rightarrow u^3$. Because there is only one occurrence of the predicate p in the body of each rule, condition 2 on ϕ forces $\phi(\mathtt{p}(\mathtt{XA}')) = \mathtt{p}(\mathtt{XA}'')$. The derived mapping $\dot{\phi}$ therefore has $\dot{\phi}(\mathtt{X}) = \mathtt{X}$, and $\dot{\phi}(\mathtt{A}') = \mathtt{A}''$. Because $\dot{\phi}(\mathtt{A}') =$

$\mathtt{A}''$, we must have $\phi(\mathtt{e}(\mathtt{A}'\mathtt{A})) = \mathtt{e}(\mathtt{A}''\mathtt{A}')$, since the latter is the only predicate which has $\mathtt{A}''$ in position 1. The latter equation in turn forces $\dot{\phi}(\mathtt{A}) = \mathtt{A}'$ in the derived mapping. On the other hand, condition 3 forces $\phi(\mathtt{e}(\mathtt{AY})) = \mathtt{e}(\mathtt{AY})$, because $\mathtt{e}(\mathtt{AY})$ is the only body predicate of u^3 which has the distinguished variable $\mathtt{Y}$ occurring in position 2. The derived mapping $\dot{\phi}$ consequently has $\dot{\phi}(\mathtt{A}) = \mathtt{A}$, which together with $\dot{\phi}(\mathtt{A}) = \mathtt{A}'$, above, amounts to a contradiction: $\dot{\phi}$ is not single-valued. We conclude that no such containment mapping ϕ exists.

2.2.2 The query containment theorem

Now let s and t be arbitrary Datalog rules with the same head predicate p. We say s *contains* t (and write $t \preceq s$) if, for every initial database D over the predicates appearing in the bodies of s and t, the set of p-facts which can be derived by one application of the rule t is a subset of the set of p-facts which can be derived by one application of the rule s. Note that we do not consider p-facts that are derivable only by recursive expansion of the rules s and t in this definition.

The following theorem can be traced to [ChMe77], although the present proof follows the argument in [Ullm88] closely.

Theorem 2.1 (The query containment theorem) *Let s and t be two Datalog rules. Then s contains t if and only if there exists a containment mapping $\phi : s \rightarrow t$.*

Proof Suppose first that there is a containment mapping $\phi : s \rightarrow t$, and let $p = p(x_1 \cdots x_n)$ be an arbitrary p-fact which is t-derivable over a particular database D. We need to show that p is also s-derivable over D. We proceed as we did with the s-derivation 2.7 of the previous section: for each variable v (distinguished or nondistinguished) occurring in s, we bind this value to $\dot{\phi}(v)$. Because $\dot{\phi}$ is a function, this binding scheme is consistent; because ϕ maps the head predicate of s to the head predicate of t, the p-fact $p = p(x_1 \cdots x_n)$ appears at the head of the rule s. Finally, each resulting bound predicate in the body of s is easily seen to be a fact of D. Thus the effect of binding each variable v of s to $\dot{\phi}(v)$ is to produce an s-derivation of the p-fact p. We have proven the theorem in one direction.

Conversely, suppose that $t \preceq s$. We need to construct a containment mapping $\phi : s \rightarrow t$. To each distinct variable v occurring in t, assign a unique ground symbol $\tau(v)$. Then for each predicate q in the body of t, create a q-fact by substituting $\tau(v)$ for each variable v in q. Let D be the database constructed in this way. If the head predicate of t is $\mathtt{p}(\mathtt{X}_1 \cdots \mathtt{X}_n)$,

then the p-fact $p = \mathrm{p}(\tau(\mathrm{X}_1)\cdots\tau(\mathrm{X}_n))$ is t-derivable over D. Because $t \preceq s$, we see that p is also s-derivable over D; moreover, the facts used in any such s-derivation necessarily come from D. We now have in hand an s-derivation and a t-derivation of the p-fact p, each over the database D. Here's how to construct a containment mapping $\phi : s \to t$. First, we map the head predicate of s to the head predicate of t, as we must. Then, for each body predicate q in s, we examine the corresponding bound predicate used in the s-derivation of p, above. This bound predicate is necessarily a fact of D, which in turn arose from some body predicate r of t. We then define $\phi(q) = r$. (If multiple choices for r exist, the choice may be made arbitrarily). That ϕ is indeed a containment mapping follows directly from the uniqueness of the values $\tau(v)$. ∎

2.2.3 Syntactic expansion

Up to this point, we have been a bit imprecise about how linear Datalog rules are recursively expanded to form new rules (such as the rule "*st*" in equation 2.6, above). In the present section, we shall make the recursive expansion process mathematically precise. Our method is basically a generalization of the "ExpandRule" procedure described by Naughton [Naug86a], although our "subgoal substitutions" are perhaps more reminiscent of the "substitution graphs" of [JAN87] than they are of Naughton's "A/V Graphs." In any case, where both [Naug86a] and [JAN87] are more concerned with expanding a *single* recursive rule, here we formalize the process by which an *arbitrary* set of linear recursive rules each with the same head predicate p can be expanded in arbitrary orders. There is no mystery in our method, however: it is little more than a formalization of how one carries out recursive expansion by hand. We shall call our method *syntactic expansion*.

To do an expansion of a linear recursive rule u by a second such rule v correctly, we must focus on two issues. First, we must identify the way in which some variables of u are passed down to v through the recursive subgoal p of u. Second, we must have a method to distinguish between nondistinguished variables that arise newly in the expansion of v from their counterparts in u. For example, consider the two rules u and v defined as follows:

$$u : \mathrm{p}(\mathrm{X}_1\mathrm{X}_2\mathrm{X}_3\mathrm{X}_4) \ :- \ \mathrm{p}(\mathrm{X}_2\mathrm{X}_1\mathrm{A}\mathrm{X}_4) \ \&\ \mathrm{e}(\mathrm{X}_3\mathrm{A}\mathrm{B}) \tag{2.8}$$

$$v : \mathrm{p}(\mathrm{X}_1\mathrm{X}_2\mathrm{X}_3\mathrm{X}_4) \ :- \ \mathrm{p}(\mathrm{A}\mathrm{X}_1\mathrm{X}_3\mathrm{B}) \ \&\ \mathrm{f}(\mathrm{X}_4\mathrm{X}_2\mathrm{C}\mathrm{X}_3). \tag{2.9}$$

To expand u by v, we first form *subgoal substitutions* corresponding to u and v. The subgoal substitution corresponding to a rule is obtained by positionally pairing each variable

in the head of the rule with the corresponding variable in the recursive p subgoal in the body of the rule. Thus for the rule u we have the subgoal substitution

$$\langle X_1|X_2,\ X_2|X_1,\ X_3|A,\ X_4|X_4\rangle_u, \tag{2.10}$$

which we read as "In u, X_1 is replaced by X_2, X_2 is replaced by X_1, X_3 is replaced by A, and X_4 is replaced by X_4." Similarly, the subgoal substitution for v is written

$$\langle X_1|A,\ X_2|X_1,\ X_3|X_3,\ X_4|B\rangle_v.$$

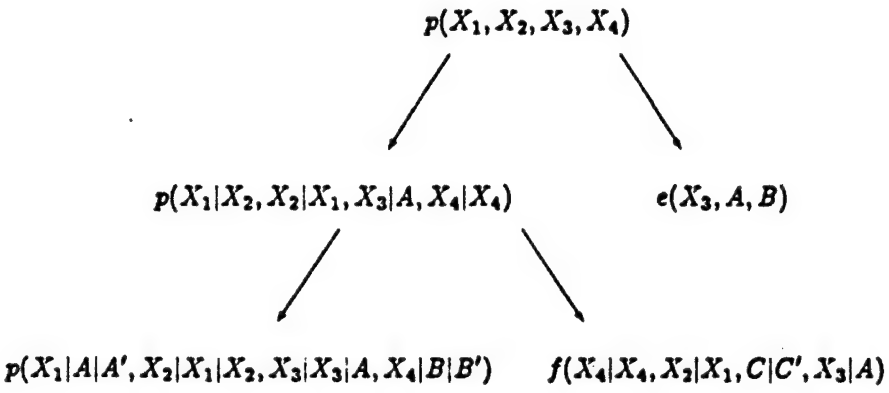
Next, we must *compose* the respective subgoal substitutions to obtain a third substitution

$$\langle X_1|A|A',\ X_2|X_1|X_2,\ X_3|X_3|A,\ X_4|B|B'\rangle_{uv}. \tag{2.11}$$

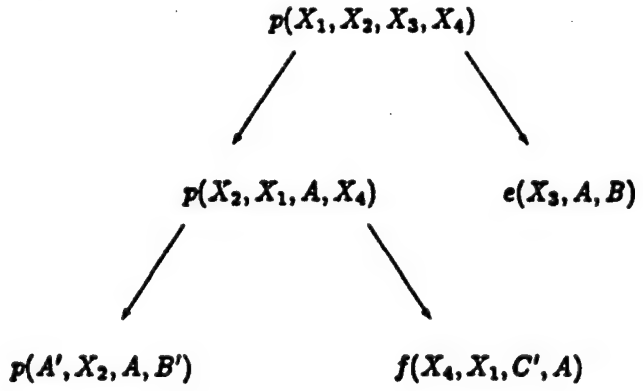
There are several important points to be made. First, observe that to form the substitution for uv, we have begun with the substitution for v, and have followed it with the substitution for u (not the reverse order). Also, note that the nondistinguished variables A and B of v have been replaced by the new formal symbols A' and B', respectively, in the u-half of the uv subgoal substitution. These substitutions are necessary in order to avoid confusion between the A's and B's of u and those of v. More formally, we could have written the symbols $\langle A|A'\rangle_u$ and $\langle B|B'\rangle_u$ as part of the u-subgoal substitution 2.10, above, to stand for these two substitutions, but because such substitutions are inherent in recursive expansion by u regardless of the particular rule u, we call them *implicit substitutions*, and do not write them down in subgoal substitution expressions. In more general compositions of subgoal substitutions, we must replace A' by a second formal symbol A'', and A'' by A''', and so on, so that in general, every linear recursive rule r has an infinite set of implicit substitutions of the form $\langle A^{(i)}|A^{(i+1)}\rangle_r$ corresponding to every variable A occurring in r that doesn't appear in the head of the rule.

A symbol like $\langle X_2|X_1|X_2\rangle_{uv}$, which is part of the the subgoal substitution expression 2.11 above, is called a *substitution chain*, and we read it as "In uv, X_2 is replaced by X_1, which is replaced in turn by X_2." We should think of a substitution chain as no more than a composition of mappings, so that the substitution chain $\langle X_2|X_1|X_2\rangle_{uv}$ is really just a more involved way of writing $\langle X_2|X_2\rangle_{uv}$. However, when we are proving things about recursive expansion, it sometimes will be more convenient to work with substitution chains in their unsimplified forms.

Now, how do we use these subgoal substitutions to expand rules? It is easiest if we think of an abstract *expansion tree* to which the appropriate variable substitutions are applied at every node. For example, to expand u by v, we first write down an expansion tree of the form



Note that to the EDB subgoals e and f, we have applied only the variable substitutions corresponding to the expansions along the path leading from these subgoals to the root of the tree, so that e has no subgoal substitutions applied to its variables, while f has only the u-substitution applied to its variables. Next, we simplify the substitution chains at every variable position in every node of the tree by replacing each substitution chain with its terminal symbol, obtaining the simplified expansion tree



```
begin
   Write down the rule r = r_m.
   for i = (m-1) down to 1 do begin
       (1) Apply the r_i subgoal substitution
           to all variables in the body of r.
       (2) Append the nonrecursive subgoals of r_i to r.
       {* Loop invariant: r = r_i ... r_m *}
       end
   end
```

Figure 2.1: Deriving $r = r_1 r_2 \cdots r_m$ by syntactic expansion.

The rule expansion uv itself can now be read off from the root and leaves of the tree: we have

$$uv : \mathtt{p(X_1X_2X_3X_4)} \ :- \ \mathtt{p(A'X_2AB')} \ \&\ \mathtt{f(X_4X_1C'A)} \ \&\ \mathtt{e(X_3AB)}. \tag{2.12}$$

Having established a method for expanding one rule r_1 by a second one r_2, we turn next to the general problem of syntactically expanding an arbitrary set of linear recursive rules in a specified order. Suppose then that we are interested in expanding $r_1 r_2 \cdots r_m$. Because subgoal substitutions compose "in the reverse order," we may apply the simple algorithm in Figure 2.1.

Example 2.2 As an illustration of syntactic expansion, suppose we take the three rules

$$\begin{aligned}
r_1 &: \mathtt{p(X_1X_2X_3)} \ :- \ \mathtt{p(X_2X_1A)} \ \&\ \mathtt{e(BX_3)} \\
r_2 &: \mathtt{p(X_1X_2X_3)} \ :- \ \mathtt{p(AX_1B)} \ \&\ \mathtt{f(X_2CX_3)} \\
r_3 &: \mathtt{p(X_1X_2X_3)} \ :- \ \mathtt{p(X_2AX_1)} \ \&\ \mathtt{g(X_3)},
\end{aligned}$$

for which we wish to form the rule expansion $r_1 r_2 r_1 r_3$. Following the algorithm, we begin by writing down the last rule of the desired expansion, $r = r_3$.

$$r : \mathtt{p(X_1X_2X_3)} \ :- \ \mathtt{p(X_2AX_1)} \ \&\ \mathtt{g(X_3)}.$$

Entering the for loop, we apply the r_1 subgoal substitution $\langle X_1|X_2, X_2|X_1, X_3|A\rangle_{r_1}$ to all the variables in the body of r, obtaining the working expression

$$r : \mathtt{p(X_1X_2X_3)} \ :- \ \mathtt{p(X_1A'X_2)} \ \&\ \mathtt{g(A)},$$

and then we append the nonrecursive subgoal of r_1 to r, to obtain the rule

$$r_1r_3 : \mathrm{p}(\mathrm{X}_1\mathrm{X}_2\mathrm{X}_3) \ :- \ \mathrm{p}(\mathrm{X}_1\mathrm{A}'\mathrm{X}_2) \ \& \ \mathrm{g}(\mathrm{A}) \ \& \ \mathrm{e}(\mathrm{BX}_3).$$

So far, we have passed through the for loop once.

On the next pass through the loop, the r_2 subgoal substitution $\langle X_1|A, X_2|X_1, X_3|B\rangle_{r_2}$ is applied to all the variables of $r = r_1r_3$, and then the nonrecursive subgoal $\mathrm{f}(\mathrm{X}_2\mathrm{CX}_3)$ of r_2 is appended to r. We obtain the rule

$$r_2r_1r_3 : \mathrm{p}(\mathrm{X}_1\mathrm{X}_2\mathrm{X}_3) \ :- \ \mathrm{p}(\mathrm{AA}''\mathrm{X}_1) \ \& \ \mathrm{g}(\mathrm{A}') \ \& \ \mathrm{e}(\mathrm{B}'\mathrm{B}) \ \& \ \mathrm{f}(\mathrm{X}_2\mathrm{CX}_3).$$

There is only one more pass through the loop to be completed, corresponding to the r_1 occurrence at the beginning of desired expansion $r_1r_2r_1r_3$. Passing through steps (1) and (2) of the syntactic expansion algorithm a final time, the desired rule expansion is seen to be

$$r_1r_2r_1r_3 : \mathrm{p}(\mathrm{X}_1\mathrm{X}_2\mathrm{X}_3) \ :- \ \mathrm{p}(\mathrm{A}'\mathrm{A}'''\mathrm{X}_2) \ \& \ \mathrm{g}(\mathrm{A}'') \ \& \ \mathrm{e}(\mathrm{B}''\mathrm{B}') \ \& \ \mathrm{f}(\mathrm{X}_1\mathrm{C}'\mathrm{A}) \ \& \ \mathrm{e}(\mathrm{BX}_3).$$

■

2.2.4 The semigroup $\mathcal{J}(S)$

Let S be an arbitrary nonempty finite set of linear recursive Datalog rules, each with the same head predicate p, and let $\dot{\mathcal{J}}(S)$ be the (infinite) set of all finite-depth top-down syntactic expansions of the rules of S. If we give names $u_1, \ldots, u_n$ to the rules in S, then we can think of the elements of $\dot{\mathcal{J}}(S)$ as in one-to-one correspondence with the set U^* of all finite words in the alphabet $U = \{u_1, \ldots, u_n\}$. When two top-down expansions s and t in $\dot{\mathcal{J}}(S)$ simultaneously stand in the relations $s \preceq t$ and $t \preceq s$, then we write $s \simeq t$, and say "s is equivalent to t."

Lemma 2.3 *The relation $\simeq$ defines an equivalence relation on $\dot{\mathcal{J}}(S)$.*

Proof To see that $\simeq$ is reflexive, it suffices to note that we always have the trivial containment $u \preceq u$ for any rule u. Also, by definition $\simeq$ is symmetric. To show that $\simeq$ is transitive, suppose that we have three rules s, t, and u, and that $s \simeq t$ and $t \simeq u$. We want to show that $s \simeq u$. Let $\phi_1 : s \to t$ and $\phi_2 : t \to u$ be containment mappings.

Then we claim that the composite mapping $\phi = \phi_2\phi_1$ defines a third containment mapping $\phi : s \rightarrow u$. The verifications of the four necessary conditions on ϕ are easily checked. First, ϕ must map the head predicate of s to the head predicate of u, since ϕ_1 maps the head predicate of s to the head predicate of t, and ϕ_2 maps the head predicate of t to the head predicate of u. Similarly, ϕ must map each body predicate of s to a body predicate of u with the same name, since ϕ_1 and ϕ_2 individually have this property. Finally, if q_1 and q_2 are predicates of s with identical (distinguished/nondistinguished) variables at positions i and j, respectively, then $\phi(q_1)$ and $\phi(q_2)$ must also have identical (distinguished/distinguished or nondistinguished) variables at positions i and j, respectively, again precisely because ϕ_1 and ϕ_2 individually have these properties.

We have shown therefore that $s \preceq t$ and $t \preceq u$ together imply $s \preceq u$. Since the proof of the opposite containment is completely symmetric, we omit it, and conclude that $s \simeq u$, as required.

∎

We are now ready to define the idea of a *rule expansion semigroup* $\mathcal{J}(S)$. The elements of $\mathcal{J}(S)$ are the equivalence classes of the relation $\simeq$ on $\dot{\mathcal{J}}(S)$. The "multiplication" in $\mathcal{J}(S)$ is defined by rule expansion: to form the product J_1J_2 of two equivalence classes $J_1, J_2 \in \mathcal{J}(S)$, one first takes in $\dot{\mathcal{J}}(S)$ two representative rule expansions $j_1 \in J_1$ and $j_2 \in J_2$. After finding the equivalence class $J \in \mathcal{J}(S)$ to which the rule $j = j_1j_2$ in $\dot{\mathcal{J}}(S)$ belongs, one finally defines $J_1J_2 = J$ in $\mathcal{J}(S)$.

There are several points to be checked. First, we must check that this process is well defined; i.e., we must verify that the equivalence class to which j belongs is independent of the choice of the representatives j_1 and j_2.

Lemma 2.4 (The splicing lemma) *Let s, t, u, and v be four linear recursive Datalog rules each with the same head predicate p. Suppose that $s \simeq t$ and $u \simeq v$. Then $su \simeq tv$.*

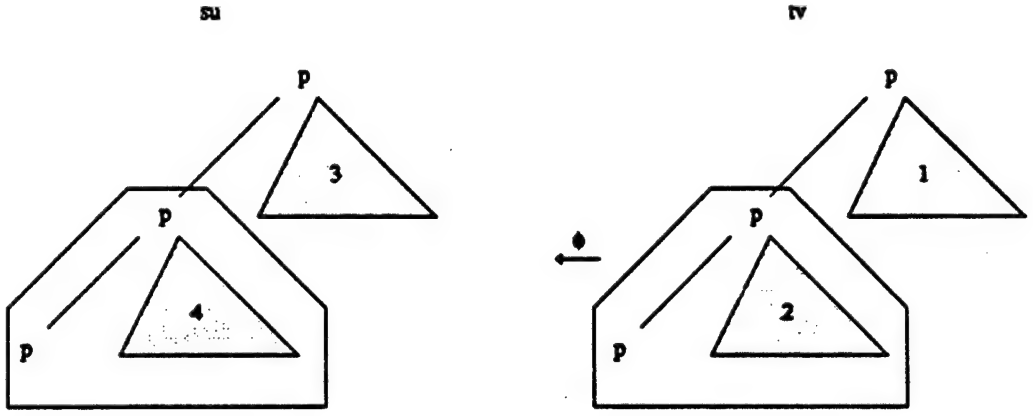
Proof In light of the query containment theorem (Theorem 2.1), it is possible to prove the splicing lemma by either of two methods. One can either work nonconstructively with the abstract containment and monotonicity properties of Datalog rules (see below), or alternatively one can produce a constructive proof by working with containment mappings and the syntactic expansion of rules. It turns out that the former argument is simpler, and we shall present it below. But because the constructive proof yields some additional

information about the structure of containment mappings $\phi : su \rightarrow tv$ that we shall exploit in Chapter 4, we shall present the latter argument first.

By assumption, there are containment mappings $\phi_1 : t \rightarrow s$ and $\phi_2 : v \rightarrow u$. We shall show that these two mappings can be "spliced together" to obtain a third containment mapping $\phi : tv \rightarrow su$. In other words, we will have shown that the conditions $s \preceq t$ and $u \preceq v$ together imply $su \preceq tv$. Because the proof of the opposite containment is completely symmetric, we omit it.

Without loss of generality, we shall assume that the linear recursive predicate p has arity $\alpha \geq 1$, and that the head predicates of t, s, u, and v are all $p(X_1 X_2 \cdots X_\alpha)$.

In the figure below, we illustrate the two top-down expansions su and tv. As indicated, the rule su (respectively, tv) is obtained from s (respectively, t) by expanding its p-subgoal by the rule u (respectively, v). We have used triangles to indicate the collection of non-recursive predicates in a given rule expansion, and we have omitted the variables of the p-predicates for clarity.



The boxes we have put around the lower halves of each rule expansion are meant to draw particular attention to the following point, which follows directly from the definition of recursive expansion: what appears in the box in su is precisely the rule u, except that the s-subgoal substitution has been applied to all of its variables, while what appears in the box in tv is precisely the rule v, except that the t-subgoal substitution has been applied to all of its variables.

Also, because ϕ_1 is a containment mapping, the following two properties are also immediate:

I. If $\langle X_i|X_j\rangle_t$, then $\langle X_i|X_j\rangle_s$.

II. If $\langle X_i|A\rangle_t$, $\langle X_j|A\rangle_t$, and $\langle X_i|V\rangle_s$, then $\langle X_j|V\rangle_s$.

In property II., the variable A is an arbitrary nondistinguished variable, while V is allowed to be any variable (distinguished or nondistinguished).

In order to construct explicitly a containment mapping $\phi : tv \rightarrow su$, the containment mappings ϕ_1 and ϕ_2 may be spliced together in the following way: we make ϕ agree with ϕ_1 on the nonrecursive t-subgoals in tv, and we make ϕ agree with ϕ_2 on the p-subgoal and on the nonrecursive v-subgoals of tv. In other words, ϕ maps the tv-predicates in triangle 1 of the figure to the su-predicates of triangle 3 in the figure exactly as ϕ_1 maps these predicates from t to s; also ϕ maps the predicates in triangle 2 to the predicates in triangle 4 exactly as ϕ_2 maps these predicates from v to u. Finally, ϕ maps the terminal p-predicate at the lower left-hand corner of tv to the corresponding p-predicate at the lower left-hand corner of su. Of course, ϕ is also made to map the head predicate of tv to the head predicate of su.

We claim that the mapping ϕ so constructed is a containment mapping from tv to su. We shall check each of the four containment axioms in turn.

Note first that by definition, ϕ maps the head predicate of tv to the head predicate of su, and these predicates have the same name. So we have verified condition 1 on ϕ. Second, ϕ must map the body tv-predicates to body su-predicates of the same name, because ϕ_1 and ϕ_2 do so individually. We have therefore verified condition 2 on ϕ. Only conditions 3 and 4 remain to be checked.

We first turn to the simpler condition 3. Because we have assumed that each of the four rules s, t, u and v has head predicate $p(X_1X_2\cdots X_a)$, verifying condition 3 on ϕ comes down to checking that if X_i is any distinguished variable in tv, then $\phi(X_i) = X_i$ in su. The verification is trivial for distinguished variables that occur in triangle 1, because ϕ_1 is a containment mapping and no subgoal substitutions are applied to any of these variables when t is expanded by v. Also, because ϕ_2 is a containment mapping and we have property I., above, the verification is also immediate for distinguished variables of tv that occur either in triangle 2 or in the recursive p-subgoal of tv.

To verify condition 4 on ϕ, suppose that q_1 and q_2 are predicates of tv with the same nondistinguished variable appearing in positions i and j, respectively. We must verify that $\phi(q_1)$ and $\phi(q_2)$ have identical variables (distinguished or nondistinguished) at positions i and j, respectively, in su. The only case to be considered which is not an immediate consequence of property II and the fact that ϕ_1 and ϕ_2 are containment mappings occurs when q_1 is a predicate of triangle 1, and q_2 is either the recursive p subgoal or a predicate of triangle 2. In this case, q_1 and q_2 share some nondistinguished variable A that is necessarily passed through the intermediate p-subgoal of tv, and we must verify that ϕ defines the value $\dot{\phi}(A)$ uniquely.

Suppose then that $\dot{\phi}_1(A)$ is some variable V in the rule s. We claim that that $\dot{\phi}(A) = V$ also.

In proof, note that in order for the variable A to passed to the expansion by v in tv, the rule t must have a subgoal substitution of the form $\langle X_k|A\rangle_t$ for some k, where X_k also appears at position j in q_2 (in v). Also, because $\dot{\phi}_1(A) = V$ and ϕ_1 is a containment mapping, the rule s has the subgoal substitution $\langle X_k|V\rangle_s$. Now because ϕ_2 is a containment mapping, u must have the variable X_k at position j in $\phi_2(q_2)$, and when s is expanded by u, the last X_k is replaced by V, so that $\dot{\phi}(A) = V$, as we claimed.

Therefore $su \preceq tv$, as required.

∎

The splicing lemma can be proved nonconstructively by referring to the following *monotonicity* property of Datalog rules: *if firing a Datalog rule r over a database D returns a set of answers A, and D' is a database containing D, then r returns an answer set A' containing A when r is fired over D'*. The monotonicity property for Datalog rules follows from the monotonicity properties of the basic relational algebra operations union, select, project, and product, and a proof can be found in [Ullm88].

With the monotonicity property in hand, we now can prove the splicing lemma as follows. Let D be a database and suppose that $s \succeq t$ and $u \succeq v$. If p_0 is a p-fact returned by the rule tv over D, then every intermediate p-fact p_1 at the head of v in a tv proof tree for p_0 can also be derived at the head of u in some su proof tree. Therefore, whatever p-facts $P_{v,D}$ can reach the p-subgoal of t through v in tv proof trees can also reach the p-subgoal of s through u in su proof trees, and these respective databases of p-facts $P_{v,D}$ and $P_{u,D}$ stand in the relation $P_{v,D} \subseteq P_{u,D}$. Now by monotonicity and the fact that $s \succeq t$, we conclude that p_0 is also a p-fact returned by the rule su over D. So $su \succeq tv$, as required.

Recall that a *semigroup* is a nonempty set $\mathcal{G}$ together with an associative binary operation on $\mathcal{G}$ [Hu74]. To verify that the set $\mathcal{J}(S)$, together with the binary operation of syntactic expansion, is indeed a semigroup, we must check that the operation of top-down recursive expansion of linear Datalog rules is *associative*; i.e., for any three rules r_1, r_2, and r_3, the rule $(r_1r_2)r_3$ obtained by expanding r_1 by r_2, and then expanding the result by r_3, is equivalent to the rule $r_1(r_2r_3)$ obtained by expanding r_1 by the rule obtained by expanding r_2 by r_3:

Lemma 2.5 *Suppose r_1, r_2, and r_3 are linear recursive Datalog rules each with the same head predicate p. Then $(r_1r_2)r_3 \simeq r_1(r_2r_3)$.*

Proof It suffices to check that the operation of subgoal substitution composition is itself an associative operation. To do this, suppose that $\langle U|V\rangle_{r_1}$, $\langle T|U\rangle_{r_2}$, and $\langle S|T\rangle_{r_3}$ are three substitutions of r_1, r_2 and r_3, respectively. Then $\langle S|T|U|V\rangle_{r_1r_2r_3} = \langle S|V\rangle_{r_1r_2r_3}$ regardless of whether we think of the composition as accomplished according to the "$(r_1r_2)r_3$ grouping" $S|(T|U|V)$, or alternatively according to the "$r_1(r_2r_3)$ grouping" $(S|T|U)|V$, so that the variable patterns in the subgoals of $r_1r_2r_3$ do not depend on the order in which we expand these three rules. ∎

Summing up these results, we can say that while the relation $\succeq$ on the set $\dot{\mathcal{J}}(S)$ of recursive expansions of S is a *quasiorder* (i.e., reflexive and transitive), $\mathcal{J}(S)$ itself is a *partially ordered semigroup* with order relation $\succeq$.

Here are some simple examples of rule expansion semigroups.

Example 2.6 Let $S = \{s\}$ consist of the single rule

$$s : \mathrm{p(XY)} \ :- \ \mathrm{p(YX)}. \tag{2.13}$$

The set $\dot{\mathcal{J}}(S) = \{s, s^2, s^3, \ldots\}$ consists of all top-down expansions of the rule s, and is infinite. However, it is easy to see that we have $s^{2i} \simeq s^{2j}$ and $s^{2i-1} \simeq s^{2j-1}$ every pair of positive integers i and j, so that $\dot{\mathcal{J}}(S)$ partitions into exactly two equivalences classes, namely

$$J_1 = \{s^{2i} \mid i \geq 1\},$$

and

$$J_2 = \{s^{2i-1} \mid i \geq 1\}.$$

The rule expansion semigroup $\mathcal{J}(S) = \{J_1, J_2\}$ is isomorphic to the additive group Z_2 of integers taken modulo 2. ∎

Example 2.7 Let $S = \{u\}$ consist of the transitive closure rule 2.1 of Section 2.1, above:

$$u : \mathrm{p}(\mathtt{XY}) \ :- \ \mathrm{p}(\mathtt{XA}) \ \& \ \mathrm{e}(\mathtt{AY}).$$

In this example, the relation $\simeq$ on the set $\dot{\mathcal{J}}(S) = \{u, u^2, u^3, \ldots\}$ of all top-down expansions of the rule u is trivial: every element of $\dot{\mathcal{J}}(S)$ defines its own singleton equivalence class. (For a proof, see Example 3.3 below). Thus, we have a one-to-one correspondence between $\dot{\mathcal{J}}(S)$ and $\mathcal{J}(S)$, and we can identify every equivalence class $J_i \in \mathcal{J}(S)$ with the unique rule expansion u^i which defines it. The rule expansion semigroup $\mathcal{J}(S)$ is therefore isomorphic to the free semigroup $U^+ = \{u, uu, uuu, \ldots\}$ generated by the formal symbol u. ∎

Example 2.8 Let $S = \{u, v\}$ consist of the two rules

$$u : \mathrm{p}(\mathtt{XY}) \ :- \ \mathrm{p}(\mathtt{XA}) \ \& \ \mathrm{e}(\mathtt{AY})$$
$$v : \mathrm{p}(\mathtt{XY}) \ :- \ \mathrm{p}(\mathtt{AY}) \ \& \ \mathrm{e}(\mathtt{XA}).$$

The elements of $\dot{\mathcal{J}}(S)$ are in one-to-one correspondence with the set U^+ of all finite words over the formal alphabet $U = \{u, v\}$. It's not hard to check, however, that we have $uv \simeq vu$. We have

$$uv : \mathrm{p}(\mathtt{XY}) \ :- \ \mathrm{p}(\mathtt{A'A}) \ \& \ \mathrm{e}(\mathtt{XA'}) \ \& \ \mathrm{e}(\mathtt{AY})$$

and

$$vu : \mathrm{p}(\mathtt{XY}) \ :- \ \mathrm{p}(\mathtt{AA'}) \ \& \ \mathrm{e}(\mathtt{A'Y}) \ \& \ \mathrm{e}(\mathtt{XA}),$$

and we can show there are containment mappings both ways between these two rules. In fact, the function $\phi : uv \rightarrow vu$ given by

$$\begin{aligned}
\phi(\mathrm{p}(\mathtt{XY})) &= \mathrm{p}(\mathtt{XY}) \\
\phi(\mathrm{p}(\mathtt{A'A})) &= \mathrm{p}(\mathtt{AA'}) \\
\phi(\mathrm{e}(\mathtt{XA'})) &= \mathrm{e}(\mathtt{XA}) \\
\phi(\mathrm{e}(\mathtt{AY})) &= \mathrm{e}(\mathtt{A'Y}).
\end{aligned}$$

is a containment mapping whose inverse ϕ^{-1} not only exists, but also is a second containment mapping $\phi^{-1} : vu \rightarrow uv$. More generally, one finds that two top-down expansions $\dot{J}_1$ and $\dot{J}_2$ in $\dot{\mathcal{J}}(S)$ stand in the relation $\dot{J}_1 \simeq \dot{J}_2$ if and only if they involve the same number of expansions by the rules u and v individually. The rule expansion semigroup $\mathcal{J}(S)$ is therefore isomorphic to the *free commutative semigroup*

$$\{u^i v^j \mid i, j \geq 1, \text{ not both } i = j = 0\}, \tag{2.14}$$

on two letters u and v. In this semigroup, multiplying two elements $u^{i_1}v^{j_1}$ and $u^{i_2}v^{j_2}$ yields the element $u^{i_1+i_2}v^{j_1+j_2}$. ∎

2.3 Basis rules

So far, we have ignored the basis rules of a program in order to concentrate on the multiplicative structure of its recursive rules. Unfortunately, we pay a price for this simplification when we come to study the query containment properties of a program that has its recursive rules instantiated by nonrecursive basis rules. It is not always enough to simply study the query containments on a program's set of recursive top-down expansions $\dot{\mathcal{J}}(S)$. The difficulty arises because the application of basis rules of a program may introduce new query containments that are not apparent from an inspection of its recursive rules only.

Example 2.9 Consider the program $\Pi = \{b, u\}$ defined by

$$b : \mathrm{p(X)} \ :- \ \mathrm{e(XA)} \tag{2.15}$$

$$u : \mathrm{p(X)} \ :- \ \mathrm{p(A)} \ \& \ \mathrm{e(XA)}. \tag{2.16}$$

First, we focus only on the rule expansion semigroup $\mathcal{J}(S)$ generated by the recursive rule set $S = \{u\}$. We find that just as in Example 2.7 above, $\mathcal{J}(S)$ is isomorphic to the free semigroup $\mathrm{U}^+ = \{\mathrm{u}, \mathrm{uu}, \mathrm{uuu}, \ldots\}$ generated by the formal symbol u. In fact, slightly more is true in the present example and in Example 2.7. Not only is there no relation $u^i \simeq u^j$ between top-down expansions unless $i = j$, but there is also not even a query containment $u^i \preceq u^j$ unless $i = j$.

However, to stop here would leave us with an erroneous picture of the query containments in Π. The actual relation p that Π defines is given by the infinite collection of conjunctive

queries

$$\begin{aligned} b &: \mathtt{p(X)} \ :- \ \mathtt{e(XA)}, \\ u &: \mathtt{p(X)} \ :- \ \mathtt{e(AA')} \ \& \ \mathtt{e(XA)}, \\ u^2 &: \mathtt{p(X)} \ :- \ \mathtt{e(A'A'')} \ \& \ \mathtt{e(AA')} \ \& \ \mathtt{e(XA)}, \end{aligned}$$

together with the general query for every $i \geq 3$,

$$\hat{u}^i : \mathtt{p(X)} \ :- \ \mathtt{e(A^{(i-1)}A^{(i)})} \ \& \ \mathtt{e(A^{(i-2)}A^{(i-1)})} \ \& \ \cdots \ \& \ \mathtt{e(AA')} \ \& \ \mathtt{e(XA)}. \quad (2.17)$$

Here, we have used the notation $\hat{u}^i$ to stand for the rule u^i with the basis rule b substituted for its p-subgoal. We see that in Π itself, we have the query containments $\hat{u}^i \preceq b$ for every $i \geq 1$: the required containment mapping $\phi_i : b \rightarrow \hat{u}^i$ is given by defining $\phi_i(\mathtt{p(X)}) = \mathtt{p(X)}$ and $\phi_i(\mathtt{e(XA)}) = \mathtt{e(XA)}$ for every $i \geq 1$. The consequences for efficient query processing of the program Π are drastic: to compute the relation p, we may apply the basis rule once, and then stop. The program Π is therefore *bounded* in the sense of Naughton and Sagiv [NaSa87]. ∎

The boundedness of the program Π in Example 2.9 above depends upon the specific interaction between its basis and recursive rules. In particular, if we substitute a different rule for the basis rule b, we may easily change Π into an (inequivalent) *unbounded* program (i.e. a program for which there is no finite depth N for which every expansion of depth $k > N$ is contained in some expansion of depth $< N$.) For example, one can easily check that the program $\Pi' = \{b', u\}$ defined by

$$\begin{aligned} b' &: \mathtt{p(X)} \ :- \ \mathtt{f(XA)} \\ u &: \mathtt{p(X)} \ :- \ \mathtt{p(A)} \ \& \ \mathtt{e(XA)}. \end{aligned}$$

is unbounded.

We shall have occasion to contrast a recursive rule set like $S = \{u\}$ (equation 2.16 above), which may or may not yield bounded programs depending on the particular basis rules chosen for it, with recursive rule sets S that yield bounded programs *regardless* of whatever basis rules we may select for them. We call a rule set of the latter type *uniformly bounded*, again following Naughton and Sagiv [NaSa88].

Example 2.10 The rule set $S = \{v\}$ containing the single rule

$$v : \mathtt{p(XY)} \ :- \ \mathtt{p(XA)} \ \& \ \mathtt{e(BY)},$$

is uniformly bounded. ∎

The program boundedness and uniform boundedness decision problems are known to be undecidable even for quite simple classes of programs [Abit89], [GMSV87], [Vard88]. Even so, for certain restricted classes of programs, useful algorithms have been developed for studying boundedness, and these algorithms can be fruitfully applied to optimize programs [Ioan85], [Ioan89], [NaSa87], [NaSa88], [RSUV89], [Sar89a]. In the following chapters, we shall take as our starting point the ideas of boundedness, the rule expansion semigroup, and *rule commutativity* [Ioan89], [RSUV89], and we shall hope to explain how classical ideas from mathematical semigroup theory can be used not only to codify and simplify some previous results in the query optimization literature, but also to give us new results about query optimization and point directions to new issues.

Chapter 3

Free rules

In the present chapter, we shall again restrict ourselves to the study of finite nonempty *rule sets* S whose members are *linear recursive* Datalog rules over a *single* recursive predicate p. A paradigmatic singleton rule set $S = \{r\}$ would consist of the transitive closure rule

$$r : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \mathtt{e(AX_2)}. \tag{3.1}$$

Again, in such a rule, the variables appearing in the heads of the rules are called *distinguished variables*, while those appearing only in rule bodies are called *nondistinguished variables*. It will again be our assumption throughout that if the arity of the recursive predicate p is $t \geq 1$, then the head predicate of every rule in S has t distinct distinguished variables occurring in it, so we assume that the head predicate of every rule in S is $p(X_1X_2 \cdots X_t)$.

In a more complex example, we might have a rule set $S = \{r_1, r_2, r_3\}$ that consists of three recursive rules

$$r_1 : \mathtt{p(X_1X_2X_3)} \ :- \ \mathtt{p(X_2AX_3)} \ \& \ \mathtt{s(X_1A)} \tag{3.2}$$

$$r_2 : \mathtt{p(X_1X_2X_3)} \ :- \ \mathtt{p(CX_3D)} \ \& \ \mathtt{t(BD)} \ \& \ \mathtt{s(AC)} \ \& \ \mathtt{s(X_2B)} \ \& \ \mathtt{s(X_1A)} \tag{3.3}$$

$$r_3 : \mathtt{p(X_1X_2X_3)} \ :- \ \mathtt{p(X_3CB)} \ \& \ \mathtt{s(AC)} \ \& \ \mathtt{t(X_2B)} \ \& \ \mathtt{s(X_1A)}. \tag{3.4}$$

Here, amongst the three rules of S we find the distinguished variables $\mathtt{X_1}$, $\mathtt{X_2}$, and $\mathtt{X_3}$, while the nondistinguished variables are A, B, C, and D.

Suppose $S = \{r_1, \ldots, r_k\}$ is a rule set. It is useful to pause here to collect the preliminary information contained in the previous chapter into a single paragraph. We have seen in the previous chapter that corresponding to every rule set S there is a natural *rule expansion semigroup* $G = \mathcal{J}(S)$ whose "multiplication" is defined by recursive expansion of the rules

in $\mathcal{S}$. If we associate with every finite-depth top-down recursive expansion of the rules in $\mathcal{S}$ a corresponding word w over the formal symbols $\{r_1, \ldots, r_k\}$, then when two words w_1 and w_2 simultaneously stand in the relationships $w_1 \succeq w_2$ and $w_2 \succeq w_1$, we call them *equivalent*, and we write $w_1 \simeq w_2$. The relation $\simeq$ defines an equivalence relation on the set of all finite words over the alphabet $\{r_1, \ldots, r_k\}$ which "respects recursive expansion:" for if $w_1 \succeq w_2$ and $w_3 \succeq w_4$, then we have verified (in the course of proving the splicing Lemma 2.4) that $w_1 w_3 \succeq w_2 w_4$. The set of all top-down expansions of a rule set S that are in the same equivalence class as a particular expansion w can therefore be identified with the one particular member w of that class, and $(\mathcal{J}(\mathcal{S}), \succeq)$ becomes a partially ordered semigroup. Without any loss of generality we can refer to the "element w in the rule semigroup $\mathcal{J}(\mathcal{S})$" as a shorthand for "the equivalence class to which the element w belongs in the rule semigroup $\mathcal{J}(\mathcal{S})$."

3.1 Rule factorization

Opposite to the recursive expansion of rules we may place the idea of *rule factorization*. As an illustration, we return to our rule set $\mathcal{S} = \{r_1, r_2, r_3\}$, above (equations 3.2, 3.3, and 3.4). In fact, we can simplify the structure of these three rules considerably by introducing two new rules a and b defined by

$$\begin{aligned} a &: \mathtt{p(X_2X_2X_3)} \;\; \mathtt{:-} \;\; \mathtt{p(X_2AX_3)} \;\; \mathtt{\&} \;\; \mathtt{s(X_1A)} \\ b &: \mathtt{p(X_1X_2X_3)} \;\; \mathtt{:-} \;\; \mathtt{p(X_2X_3A)} \;\; \mathtt{\&} \;\; \mathtt{t(X_1A)}, \end{aligned}$$

for which it is easily verified that

$$\begin{aligned} r_1 &\simeq a \\ r_2 &\simeq a^3 b \\ r_3 &\simeq aba. \end{aligned}$$

We may say somewhat loosely that the rules r_1, r_2, and r_3 admit a "factorization" into the simpler rules a and b. Once we have such a factorization in hand, higher-order dependencies between the rules may be won from the associative law: we have for example

$$r_2 r_1 \simeq (a^3 b)(a) \simeq (a^2)(aba) \simeq r_1^2 r_3.$$

Summarizing, we can say that the rule set S satisfies the *algebraic relationship* $r_2r_1 \simeq r_1^2r_3$, and that this algebraic relationship is *implied by* the given factorization.

More formally, let $S = \{r_1, \ldots r_k\}$ be a rule set, and suppose that w_1 and w_2 are two finite words over the formal alphabet S. Then we shall say that

$$w_1 \simeq w_2$$

is an *algebraic relationship* of S precisely when the two conjunctive queries obtained by recursively expanding the rules of S according to the words w_1 and w_2 are equivalent. A *trivial* algebraic relationship is one in which the two words w_1 and w_2 are identical. If r is the rule obtained by syntactically expanding rule r_1 by the rule r_2, then we shall say that r *admits the factorization* r_1r_2. Of course, it also follows that $r \simeq r_1r_2$ if r admits the factorization r_1r_2. If r_1 and r_2 each have at least one nonrecursive subgoal apiece, we shall call the factorization *nontrivial*; otherwise, the factorization is said to be *trivial*. The distinction between trivial and nontrivial factorizations will not be important to us until Chapter 5. If S and T are rule sets, we shall also say occasionally that S *admits a factorization over* T if every element r in S is either an element of T or alternatively r admits a factorization r_1r_2 where r_1 and r_2 are either elements of T or recursive expansions of elements of T.

3.2 Commutativity, free rules, and boundedness

Several authors [Ioan89], [RSUV89] have pointed out that the complexity of recursive query processing can be reduced when elements of a rule set satisfy a *commutative law* $r_1r_2 \simeq r_2r_1$. In the present context, we prefer to think of a commutative law as nothing more than the simplest of algebraic relationships that a given rule set S may satisfy. However, every algebraic relationship satisfied by a rule set will lead to some sort of redundancy in evaluation of its rules which a query optimizer may be able to exploit. The questions arise therefore: not only "do we have commutativity?" but more generally "what are *all* the algebraic relationships satisfied by this rule set?" In the present section, we hope to illustrate a general theory that can be applied under certain circumstances in such a setting. We need to make two definitions:

Definition 3.1 *Let S be a rule set. We say S is* weakly free *if these rules satisfy no nontrivial algebraic relationships.*

Put another way, a rule set is weakly free if it has no two distinct top-down expansions that mutually contain one another.

We shall also be interested in rule sets that are not only weakly free, but also satisfy the stronger condition that no conjunctive query in its set of top-down expansions is contained in another such query:

Definition 3.2 *Let S be a rule set. We say S is* strongly free *(or more simply,* free*) if the query containment relation $\succeq$ on the set of top-down expansions of the rules of S is trivial; i.e., no such query is contained in any other (except itself).*

It is useful at this point to give a brief catalog of examples.

Example 3.3 The simple transitive closure rule

$$u : \mathrm{p}(\mathtt{X}_1\mathtt{X}_2) \ :- \ \mathrm{p}(\mathtt{X}_1\mathtt{A}) \ \& \ \mathrm{e}(\mathtt{A}\mathtt{X}_2)$$

is strongly free. To see this, we return to the general ith recursive expansion of the rule u (compare with equation 2.2, above):

$$u^i : \mathrm{p}(\mathtt{X}_1\mathtt{X}_2) \ :- \ \mathrm{p}(\mathtt{X}_1\mathtt{A}^{(i-1)}) \ \& \ \mathrm{e}(\mathtt{A}^{(i-1)}\mathtt{A}^{(i-2)}) \ \& \ \cdots \ \& \ \mathrm{e}(\mathtt{A}'\mathtt{A}) \ \& \ \mathrm{e}(\mathtt{A}\mathtt{X}_2).$$

(We identify $\mathtt{A}^0$ with $\mathtt{A}$ so that $u^1 = u$ in our notation).

In order to prove that the rule set $S = \{u\}$ is strongly free, we must show that there can be no containment mapping $\phi : u^i \rightarrow u^j$ unless $i = j$. So suppose that $\phi : u^i \rightarrow u^j$ is a containment mapping. Then by condition 2 on ϕ, we have $\phi(\mathrm{p}(\mathtt{X}_1\mathtt{A}^{(i-1)})) = \mathrm{p}(\mathtt{X}_1\mathtt{A}^{(j-1)})$, and $\dot{\phi}(\mathtt{A}^{(i-1)}) = \mathtt{A}^{(j-1)}$. Condition 3, on the other hand, forces $\phi(\mathrm{e}(\mathtt{A}\mathtt{X}_2)) = \mathrm{e}(\mathtt{A}\mathtt{X}_2)$, so that $\dot{\phi}(\mathtt{A}) = \mathtt{A}$. To show that i must be equal to j, it is simplest to rule out the possibilities that $i > j$ and $i < j$ individually.

First suppose that $j - i = s > 0$. Then by using condition 4 on ϕ repeatedly, we find first that $\dot{\phi}(\mathtt{A}^{(i-1)}) = \mathtt{A}^{(j-1)}$ implies $\dot{\phi}(\mathtt{A}^{(i-2)}) = \mathtt{A}^{(j-2)}$, and so on, until finally we conclude that $\dot{\phi}(\mathtt{A}^0) = \mathtt{A}^{(j-i)}$. But since we already know that $\dot{\phi}(\mathtt{A}^0) = \mathtt{A}^0$, we conclude that $j - i = s = 0$, a contradiction.

Alternatively, suppose that $j - i = s < 0$. Then again by using condition 4 on ϕ repeatedly, we find first that $\dot{\phi}(\mathtt{A}^0) = \mathtt{A}^0$ implies $\dot{\phi}(\mathtt{A}^1) = \mathtt{A}^1$, and so on, until finally we conclude that $\dot{\phi}(\mathtt{A}^{(i-1)}) = \mathtt{A}^{(i-1)}$. But since we already know that $\dot{\phi}(\mathtt{A}^{(i-1)}) = \mathtt{A}^{(j-1)}$, we conclude that $i - 1 = j - 1$; i.e. $i = j$, again a contradiction.

∎

A rule set with more than one linear recursive rule may be strongly free.

Example 3.4 The rule set $S = \{c, d\}$ consisting of the two rules

$$c : p(X_1X_2) \ :- \ p(X_1A) \ \& \ e(X_2A)$$
$$d : p(X_1X_2) \ :- \ p(X_1A) \ \& \ e(AX_2)$$

is strongly free. (See definition 3.9 and Theorem 3.10, below, for more discussion). ∎

Of course, a rule set with more than one linear recursive rule need not be strongly free:

Example 3.5 The rule set consisting of the two rules

$$u : p(X_1X_2) \ :- \ p(X_1A) \ \& \ e(AX_2)$$
$$v : p(X_1X_2) \ :- \ p(AX_2) \ \& \ e(X_1A)$$

is neither strongly nor weakly free. (We have the commutative law $uv \simeq vu$—see Example 2.8). ∎

We have said that a rule set $S = \{r_1, \ldots, r_k\}$ is *uniformly bounded* if there is a constant N such that every recursive expansion w of its rules is contained in some recursive expansion of depth $\leq N$. If a rule set is not uniformly bounded, we shall simply call it *unbounded*. From the query optimization point of view, boundedness is important because it captures the idea of when a recursive rule or rule set may be replaced by an equivalent set of nonrecursive rules. We shall see below how the idea of uniform boundedness arises quite naturally in the free rule setting. First, the following theorem is immediate.

Theorem 3.6 *Strongly free rule sets are unbounded.*

Proof If S is a strongly free rule set, then it has no nontrivial containments between its recursive expansions. In particular, there is certainly no finite depth N for which every expansion is contained in some expansion of depth $< N$, so S is unbounded. ∎

More importantly, when a rule set factors into weakly free rules, then we can reduce the study of all its algebraic relationships to the study of its rule expansion semigroup:

Theorem 3.7 *Let $\mathcal{S}$ be a rule set admitting a factorization over a weakly free rule set $\mathcal{F}$. Then $\mathcal{S}$ satisfies only those algebraic relationships that are implied in the rule expansion semigroup $\mathcal{J}(\mathcal{S})$ associated with this factorization.*

Proof Suppose $\mathcal{S} = \{r_1, \ldots, r_k\}$ admits a factorization over a weakly free rule set $\mathcal{F}$. Then every rule $r_i \in \mathcal{S}$ is equivalent to some recursive expansion of rules chosen from $\mathcal{F} = \{f_1, \ldots, f_t\}$. To capture this information symbolically, we may write $r_i \simeq w_i$ for some formal word w_i over the symbols $\mathcal{F}$. Now suppose $r \simeq s$ is some algebraic relationship satisfied by the rules of $\mathcal{S}$ (here r and s are words over the alphabet $\mathcal{S}$). Replacing every occurrence of each r_i in r and s by its corresponding word w_i over $\mathcal{F}$, we find that the the resulting two words over $\mathcal{F}$ must be identical (for otherwise $\mathcal{F}$ would not be weakly free). Therefore $r \simeq s$ is a consequence of the given factorization, so our proof is complete. ∎

Even more can be said when we have a strongly free factorization:

Theorem 3.8 *Let $\mathcal{S}$ be a rule set admitting a factorization over a strongly free rule set $\mathcal{F}$. Then $\mathcal{S}$ satisfies only those algebraic relationships implied by this factorization, and the query containment relation $\succeq$ on the set of top-down expansions of the rules of $\mathcal{S}$ is an equivalence relation.*

Proof Because every strongly free rule set is also trivially weakly free, all that needs to be checked is that if s_1 and s_2 are two expansions of the rules in $\mathcal{S}$ and $s_1 \succeq s_2$, then $s_2 \succeq s_1$ is also true. Our argument proceeds exactly as in Theorem 3.7, above:

Suppose $\mathcal{S} = \{r_1, \ldots, r_k\}$ admits a factorization over a strongly free rule set $\mathcal{F}$. Then every rule $r_i \in \mathcal{S}$ is equivalent to some recursive expansion of rules chosen from $\mathcal{F} = \{f_1, \ldots, f_t\}$, and we may write $r_i \simeq w_i$ for some formal word w_i over the symbols $\mathcal{F}$, for each i. Now, if $s_1 \succeq s_2$ is some query containment between expansions of the rules in $\mathcal{S}$, then we replace each r_i in s_1 and s_2 by its corresponding word w_i over $\mathcal{F}$. Again, we find that the the resulting two words over $\mathcal{F}$ must be identical (for otherwise $\mathcal{F}$ would not be strongly free). So in fact $s_1 \simeq s_2$, and in particular, $s_2 \succeq s_1$, as we were required to show. ∎

That is to say, if a rule set $\mathcal{S} = \{r_1, \ldots, r_k\}$ can be factored into strongly free rules and we have a conjunctive query containment $w_1 \succeq w_2$ between two top-down expansions of the rules of $\mathcal{S}$, then in fact we also have $w_2 \succeq w_1$. Thus the study of *all* query containments

satisfied by the rule expansions of $\mathcal{S}$ is reduced to the study of the associated subsemigroup of the free semigroup on the symbols $\{r_1, \ldots, r_k\}$. We shall return to this idea below.

We may also think of our theorems as casting light on the following general problems:

1. Given a rule set $\mathcal{S}$, fully characterize the query containment partial ordering $\succeq$ on its set of top-down expansions. What is the computational complexity of determining whether given top-down expansions stand in the relation $w_1 \succeq w_2$?

2. Identify classes of rule sets for which the first question can be answered efficiently.

The first problem is hard: it is NP-complete to determine whether $w_1 \succeq w_2$ for general queries [ChMe77]. Yet Theorem 3.8 points us in the direction of classes of rule sets for which query containment problems admit efficient solution by semigroup techniques. For example, consider the following class of rule sets, of which the rule set in Example 3.4, above, is a representative. We need a preliminary definition.

Definition 3.9 *Let $k \geq 0$ be an integer. A* left-right transitive closure rule *is a linear, single-edb rule*

$$p(X_1X_2) :- p(X_1A^{(k)}) \ \& \ e_k \ \& \ \cdots \ \& \ e_1 \ \& \ e_0$$

where either $e_1 = e(AA^{(1)})$ or $e_1 = e(A^{(1)}A)$, and in general either $e_i = e(A^{(i-1)}A^{(i)})$ or $e_i = e(A^{(i)}A^{(i-1)})$ for $2 \leq i \leq k$; finally either $e_0 = e(X_2A)$ or $e_0 = e(AX_2)$.

For example, the rules c and d of Example 3.4 are both left-right transitive closure rules.

Theorem 3.10 *Let $\mathcal{S}$ be a rule set consisting of a finite collection of left-right transitive closure rules. Then two queries w_1 and w_2 in the set of top-down expansions of the rules of $\mathcal{S}$ can be tested for $w_1 \succeq w_2$ in linear time; moreover, whether there are any two distinct queries w_1 and w_2 such that $w_1 \succeq w_2$ can be determined in polynomial time.*

Proof Suppose $r \in \mathcal{S}$ is a left-right transitive closure rule with $k \geq 0$ nonrecursive subgoals $e_1, \ldots, e_k$ and a $(k+1)$st subgoal e_0 as in the previous definition. Then we claim r is a recursive expansion of the rules c and d of Example 3.4, above: in fact, we claim that $r \simeq \beta_{k+1}\beta_k \ldots \beta_1$, where

$$\beta_i = \begin{cases} c & \text{if } e_i = e(A^{(i-1)}A^{(i)}) \\ d & \text{if } e_i = e(A^{(i)}A^{(i-1)}) \end{cases}$$

for $1 \leq i \leq k$, and

$$\beta_{k-1} = \begin{cases} c & \text{if } e_0 = e(X_2 A) \\ d & \text{if } e_0 = e(A X_2). \end{cases}$$

The claim may be proved by induction on the integer k. First note that if $k = 0$, then r is either the rule

$$\mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \mathtt{e(X_2A)},$$

in which case $r \simeq c$, or alternatively r is the rule

$$\mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \mathtt{e(AX_2)},$$

in which case $r \simeq d$. Inductively, we now assume that the claim is valid for all left-right transitive closure rules r with k-values less than some fixed positive integer K, and consider the claim for the value $k = K$. Let r' be the rule obtained from r by deleting its rightmost EDB subgoal e_0 and decreasing all the superscripts on its nondistinguished A-variables by one (the variable A itself is to be replaced by the distinguished variable X_2). Then r' is itself a left-right transitive closure rule with one fewer EDB subgoal than r. The inductive hypothesis therefore applies, and r' is a recursive expansion of the rules c and d as in the claim statement above. Now if the occurrence of e_0 in r is $\mathtt{e(AX_2)}$, then by the definition of syntactic expansion, the rule r is obtained by recursively expanding the rule r' by the rule c, and if alternatively the occurrence of e_0 in r is $\mathtt{e(X_2A)}$, then the rule r is obtained by recursively expanding the rule r' by the rule d. We therefore have verified the claim above.

Now, because the rule set $T = \{c, d\}$ is strongly free (Example 3.4), Theorem 3.8 guarantees that a conjunctive query containment $w_1 \succeq w_2$ can be tested by writing down their respective factorizations over T, and checking whether these two words are equal. Because two words w_1 and w_2 over the formal alphabet $\{c, d\}$ can be tested for equality in time linear in $|w_1| + |w_2|$, a given containment $w_1 \succeq w_2$ also can be tested in linear time.

To test whether there exist queries w_1 and w_2 such that $w_1 \succeq w_2$, Theorem 3.8 again allows us to reduce the problem to one about words over the formal alphabet T. More precisely, we face the following problem: given a finite set W of words over the alphabet $T = \{c, d\}$, how can we recognize whether there is a single word $w \in W^*$ that admits two *distinct factorizations* into the words W? In the present context, the words W arise as factorizations of the rules in S into the rules c and d of T, and we see that there will be a nontrivial query containment $w_1 \succeq w_2$ precisely when one recursive expansion of the rules

S "decomposes" in two different ways. When there is no such word w, the set W is called a *code*. Because there are known polynomial time algorithms for recognizing codes in free semigroups (see [Speh75], and the following section, for more details), the present proof is complete.

∎

Theorem 3.10 is meant primarily as an illustration of a general technique—in fact, there are many theorems like Theorem 3.10: our ability to state them depends only on our ability to find sets of strongly free rules and describe the rule sets they "generate."

3.3 Codes and free rule set recognition

To explore these ideas a bit further and pick up some semigroup theory along the way, it is useful to consider another example. Consider the rule set $S = \{r_1, r_2, r_3\}$ given by

$$r_1 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \mathtt{e(X_2A)} \tag{3.5}$$

$$r_2 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1C)} \ \& \ \mathtt{e(CB)} \ \& \ \mathtt{e(AB)} \ \& \ \mathtt{e(AX_2)} \tag{3.6}$$

$$r_3 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1B)} \ \& \ \mathtt{e(BA)} \ \& \ \mathtt{e(X_2A)} \tag{3.7}$$

These three rules are left-right transitive closure rules. As such, they can be expressed as recursive expansions of the strongly free rules c and d of Example 3.4, above: we have

$$\begin{aligned} r_1 &\simeq c \\ r_2 &\simeq dcd \\ r_3 &\simeq cd. \end{aligned}$$

Theorems 3.8 and 3.10 allow us to make the assertion: any conjunctive query containment $w_1 \succeq w_2$ satisfied in the set of all top-down expansions of the rules in S is in fact an algebraic relationship $w_1 \simeq w_2$; moreover, this algebraic relationship is necessarily a consequence of the given factorization. Thus the question: "do the rules $\{r_1, r_2, r_3\}$, when expanded, satisfy any nontrivial conjunctive query containments $w_1 \succeq w_2$?" has been effectively reduced to the purely semigroup-theoretical question: "is the subsemigroup of the free semigroup on the formal alphabet $\{c, d\}$ generated by the set of words $\{c, dcd, cd\}$ itself a free semigroup?" The power of the techniques described above depends upon the fact that semigroup theorists

have studied such questions in some detail, and have given algorithms that may be fruitfully carried over to the query optimization setting.

Before proceeding, we answer the question asked above. Because

$$r_3 r_3 \simeq (cd)(cd) \simeq c(dcd) \simeq r_1 r_2,$$

the subsemigroup of the free semigroup on the letters c and d which is generated by c, dcd, and cd is *not* itself free. Equivalently, we can say that $\mathcal{J}(\mathcal{S})$ is not a free rule expansion semigroup, because it satisfies the algebraic relationship $r_3^2 \simeq r_1 r_2$.

The preceding discussion suggests a general question: how hard is it to recognize whether a given rule set, once factored into strongly free rules. admits any algebraic relationships? As we remarked in Theorem 3.10, above, the problem is exactly the so-called "code recognition problem" that has been completely solved in the free semigroup setting by Spehner [Speh75], who has not only given a beautiful polynomial time algorithm for the decision problem, but has also given a method that can be adapted [Lall79] to construct an abstract *presentation* of the associated rule expansion semigroup. Other algorithms for testing whether a finite set of finite words over some alphabet A form a code have been given ([Mark62], [Blu65b]), but we shall prefer Spehner's algorithm because it is the most easily explained and analyzed. Note that as a special case, we may decide commutativity in polynomial time for rule sets $\mathcal{S}$ that factor into strongly free rules. Recognizing rule commutativity for general rules appears to be difficult [Sar89b].

Here, we shall only present Spehner's algorithm in enough detail so that we can verify it can be carried out in polynomial time. In particular, we want to prove the following generalization of Theorem 3.10:

Theorem 3.11 *Suppose $\mathcal{S}$ is a rule set whose members have been factored over a strongly free rule set $\mathcal{F}$. Then the following may be decided in polynomial time: (i) recognizing commutativity; and (ii) more generally, recognizing whether there exists* any *nontrivial algebraic relationship holding amongst the rules in $\mathcal{S}$.*

Proof

Suppose $\mathcal{S} = \{r_1, \ldots, r_k\}$ admits a factorization over such a strongly free rule set $\mathcal{F}$. Then every rule $r_i \in \mathcal{S}$ is equivalent to some recursive expansion of rules chosen from a strongly free set of rules $\mathcal{F} = \{f_1, \ldots, f_t\}$, and we may write each r_i as some formal word w_i over the symbols $\mathcal{F}$.

To recognize whether two given rules r_i and r_j in $\mathcal{S}$ commute, we can simply write down their factorizations over $\mathcal{F}$ in the two orders $r_i r_j$ and $r_j r_i$, and check to see if the resulting words are identical: by Theorem 3.8, nothing more needs to be said.

To recognize whether there exists some nontrivial algebraic relationship amongst the rules in $\mathcal{S}$, we must decide whether the words w_i form a code in $\mathcal{F}^*$, the free monoid over the alphabet $\mathcal{F}$. (Recall that the free monoid over $\mathcal{F}$ is just the free semigroup over $\mathcal{F}$ with a special identity element adjoined to it. Multiplication here corresponds to string concatenation, and we can think of the identity as representing the empty word over $\mathcal{F}$.)

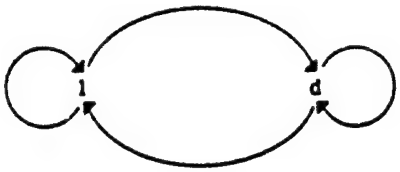
We now sketch definitions preliminary to Spehner's solution to the code recognition problem as it applies to the problem at hand. We follow Lallement [Lall79]. Let W^* be the submonoid of $\mathcal{F}^*$ generated by the words w_i. We call a pair $(u, v) \in \mathcal{F}^* \times \mathcal{F}^*$ a *W^*-pair* if there exists a $w \in W^*$ such that $w_i = uwv$ for some w_i, with $uw \neq 1$ and $wv \neq 1$. We denote the W^*-pair (u, v) by the symbol $u \rightarrow v$. A pair $(u, v) \in \mathcal{F}^* \times \mathcal{F}^*$ is called *W^*-connected* (notation $u \Rightarrow v$) if there is a sequence $u = u_0, u_1, \ldots, u_n = v$ such that $u_i \rightarrow u_{i+1}$ for every $0 \leq i < n$. Finally, we define $\Gamma(W^*)$, the *graph of* W, as the graph having as set of vertices the set $U(W) = \{w \in \mathcal{F}^* \mid w \Rightarrow 1 \text{ and } 1 \Rightarrow w\}$, and as a set of directed edges the W^*-pairs (u, v) with $u, v \in U(W)$.

For example, we return to the factorization $\{c =: w_1,\ dcd = w_2,\ cd = w_3\}$ of the rules 3.5, 3.6, and 3.7, above. To calculate the associated W^*-pairs, it is simplest if we consider each word w_i individually, and build the small table in Table 3.1.

w_i	W^*-pairs	$u \cdot w \cdot v$
c	$(1,1)$	$1 \cdot c \cdot 1$
dcd	$(1,1)$	$1 \cdot dcd \cdot 1$
	$(d,1)$	$d \cdot cd \cdot 1$
	(d,d)	$d \cdot c \cdot d$
cd	$(1,1)$	$1 \cdot cd \cdot 1$
	$(1,d)$	$1 \cdot c \cdot d$

Table 3.1: The W^*-pairs derived from the set of words $\{c, dcd, cd\}$.

Apparently $U(W') = \{1, d\}$. The graph $\Gamma(W')$ is



The result we need from Spehner can now be succinctly stated:

W' is free over the words w_i iff. the graph of W' is trivial (i.e. it is identically $1 - 1$).

In our running example, the graph $\Gamma(W')$ is not trivial, so we have here an alternative proof that the submonoid generated by c, *dcd*, and *cd* is not a code.

Finally, we turn to the time analysis of using Spehner's method to decide part (*ii*) of the theorem statement. First, note that because the monoid W' is generated by a finite set of words w_i, it is in fact a *regular* language for which we may decide the membership of any given word w in time polynomial in $|w| + \sum |w_i|$. Next, note that the equation $w_i = uwv$ entails $|w| \leq |w_i|$, so we may therefore determine whether a given possible candidate for the role of w in a decomposition $w_i = uwv$ satisfies $w \in W$ in time polynomial in $\sum |w_i|$. To generate all the W'-pairs is then also a polynomial-time problem, because we simply can try for each word w_i all the $O(|w_i|^2)$ possible decompositions $w_i = uwv$, checking in each case whether $w \in W$. The resulting graph $\Gamma(W')$ can then be scanned for connected components using simple techniques, and checked for a triviality in a final step.

∎

3.4 Presenting query containments

If a rule set S factors over a strongly free rule set, then Spehner's method can be extended to give an abstract presentation of the rule expansion semigroup $\mathcal{J}(S)$. Such a presentation is in effect a complete mathematical specification of all the redundancy encountered between the recursive expansions of the rules S, in the sense that all query containments between recursive expansions will be algebraic consequences of the given presentation.

To describe the method we need to pick up some more elementary results and definitions. Again, we follow Lallement [Lall79].

Let M be a submonoid of the free monoid A^* on an alphabet A. If we let M^+ stand for the semigroup $M - \{1\}$, then one has the following theorem [Lall79].

Theorem 3.12 *Every submonoid M of a free monoid has a unique minimal set of generators $C = M^+ - (M^+)^2$. The set C is called the* base *of M.*

Example 3.13 In the free monoid A^* on the alphabet $A = \{a, b\}$, the submonoid

$$M = \{a^2, a^3, a^4, \ldots\}$$

has as its base the set $C = \{a^2, a^3\}$. Note that M is not free over C: for example the word a^6 admits the two factorizations $a^2a^2a^2$ and a^3a^3. ∎

Restating the definition of a code in terms of bases we have

Definition 3.14 *A subset C of a free monoid A^* is called a* code *over A if C is the base of a free submonoid M of A^*.*

Returning to the notation and hypothesis of Theorem 3.11, suppose that every member r_i of a rule set S is equivalent to some recursive expansion of rules chosen from a strongly free set of rules $\mathcal{F} = \{f_1, \ldots, f_t\}$. Then as before, we may write each r_i as some formal word w_i over the symbols $\mathcal{F}$. Let $W = \bigcup_i w_i$, and let M be the monoid generated by the words w_i in the free monoid F^*. Also, we shall suppose for the moment that redundant generators w_i have been eliminated so that the words w_i are in fact a base for the submonoid W^+ of F^*. We shall return to the issue of eliminating redundant generators below.

If we construct the graph $\Gamma = \Gamma(W)$ from the W-pairs as in Theorem 3.11 and then find that the graph Γ is trivial, then it follows that the rule set S is free and that S satisfies no nontrivial query containments. It is our goal to explain how Spehner's method, in the case where the graph Γ is nontrivial, can be extended to give an abstract presentation of the semigroup $\mathcal{J}(S)$. Because the algorithm in question is essentially due to Spehner we shall omit the proof of correctness and shall satisfy ourselves with a formal description of the technique and an example.

First, we need some more definitions, which we take from [Lall79]. If (u, v) is a W-pair, then by definition there exist $z \in W$ and $y \in W^+$ such that $uyv = z$. We shall say that the pair (z, y) is *produced* by the W-pair (u, v). If $u_0, u_1, \ldots, u_n$ is a sequence of words in

F^* such that each (u_i, u_{i-1}) is a W-pair, and if (z_i, y_i) is a pair produced by (u_i, u_{i-1}) for $i = 0, 1, \ldots, n-1$, then the relation

$$z_0 y_1 z_2 \ldots = y_0 z_1 y_2 \ldots$$

is called a *relation produced* by the W-sequence $u_0, u_1, \ldots, u_n$.

A *simple W-circuit* is just such a sequence of vertices $1, u_1, u_2, \ldots, u_n$ of the graph Γ such that the pairs

$$(1, u_1), (u_1, u_2), \ldots, (u_{n-1}, u_n), (u_n, 1)$$

are W-pairs and $u_i \neq 1$ for $i = 1, \ldots, n$. We write a simple W-circuit $(1, u_1, \ldots, u_n, 1)$.

At the heart of Spehner's method is the following result [Speh75].

Theorem 3.15 *Let W^+ be a submonoid of F^+ with base W, and let $(1, x_1, x_2, \ldots, x_r, 1)$ be a simple W-circuit in the graph of Γ. Then the relation produced from this W-circuit is a relation satisfied in W^+. Moreover, every relation in W^+ is an algebraic consequence of relations produced by simple W-circuits of the graph of Γ.*

To illustrate the theorem, we return once more to the rules

$$r_1 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \mathtt{e(X_2A)}$$
$$r_2 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1C)} \ \& \ \mathtt{e(CB)} \ \& \ \mathtt{e(AB)} \ \& \ \mathtt{e(AX_2)}$$
$$r_3 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1B)} \ \& \ \mathtt{e(BA)} \ \& \ \mathtt{e(X_2A)}$$

from Section 3.3. These rules could be written as

$$r_1 \simeq c$$
$$r_2 \simeq dcd$$
$$r_3 \simeq cd,$$

where the rules c and d are taken from Example 3.4. The simple W-circuits in the graph Γ are

$$(1, d, 1)$$
$$(1, d, d, 1)$$
$$(1, d, d, d, 1)$$
$$(1, d, d, d, d, 1)$$
$$\ldots$$

for which the corresponding produced relations are

$$r_3r_3 \simeq r_1r_2$$
$$r_3r_1r_2 \simeq r_1r_2r_3$$
$$r_3r_1r_2r_3 \simeq r_1r_2r_1r_2$$
$$r_3r_1r_2r_1r_2r_3 \simeq r_1r_2r_1r_2r_1r_2$$
$$\ldots$$

Simplifying these expressions according to whether an odd or even number of d's are chosen to occur in the W-circuit, we obtain the presentation relations

$$\begin{aligned} r_3(r_1r_2)^ir_3 &\simeq (r_1r_2)^{i+1} \\ r_3(r_1r_2)^{i+1} &\simeq (r_1r_2)^{i+1}r_3, \end{aligned}$$

where i is allowed in the range $i \geq 0$.

All query containments satisfied by the rules r_1, r_2 and r_3 are consequences of the given relations above.

Chapter 4

Singleton rule sets

Given the computational power that comes along with recognizing that a given rule set factors into free rules, we may ask: how hard is it to factorize, and how hard is it to recognize free rule sets? In the present chapter, we shall begin a general study of these questions by focusing on rule sets S that consist of a single linear recursive rule r. Not surprisingly, we find points of contact between the ideas of freeness and program boundedness here. First, it is not difficult to prove:

Theorem 4.1 *Let $S = \{r\}$ be a singleton rule set. Then S is uniformly bounded if and only if there is no single nontrivial containment $r^i \succeq r^j$ with $i < j$.*

Proof A slightly weaker form of the theorem was proved by Naughton [Naug86a], but the splicing lemma allows us to give a simpler proof.

If S is uniformly bounded, then by definition it has nontrivial containments $r^i \succeq r^j$ with $i < j$.

For the converse, first note that because $r \simeq r$, we may apply the splicing lemma repeatedly to prove first that $r^{i+1} \succeq r^{j+1}$, and then inductively that $r^{i+a} \succeq r^{j+a}$ for every positive integer a.

Next, we claim that that $r^i \succeq r^{mj-(m-1)i}$ for every positive integer m.

We may prove the claim by induction on m. First, when $m = 1$, we have $r^{mj-(m-1)i} = r^j$ so the claim is valid. Now suppose the claim is valid for all values m strictly less than some positive integer t, and consider the case $m = t$. From the induction hypothesis, we have $r^i \succeq r^{(t-1)j-(t-2)i}$. Adding the constant $a = j - i$ to the superscripts on both sides, we obtain $r^j \succeq r^{tj-(t-1)i}$. Since by assumption $r^i \succeq r^j$, we can use the transitive property of

containment (Lemma 2.3) to conclude that $r^i \succeq r^{tj-(t-1)i} = r^{mj-(m-1)i}$, completing the inductive step and the proof of the claim.

To finish, we must show that there is a constant N such that for all $k \geq N$, we have $r^l \succeq r^k$ for some $l < N$; i.e., S is uniformly bounded. We shall see that the choice $N = j$ works. In proof, let $l \geq j$ be any positive integer, and use the division algorithm to write $l - i = q(j - i) + r$ for some quotient $q \geq 1$ and remainder r satisfying $0 \leq r < (j - i)$. By the claim above, $r^i \succeq r^{qj-(q-1)i}$. Adding the constant $a = r$ to the subscripts on both sides, we obtain

$$r^{i+r} \succeq r^{qj-(q-1)i+r} = r^{qj-(q-1)i+(l-i)-q(j-i)} = r^l,$$

i.e., $r^{i+r} \succeq r^l$. Note that $i + r < i + j - i = j$, as required. Since l was completely arbitrary, this completes the proof. ∎

We find therefore that recognizing whether a singleton rule set is free is closely related to the so-called "uniform boundedness problem for linear single rule programs (sirups)" [Kane88] that has attracted various researchers [Ioan85], [JAN87], [NaSa88], [Vard88]. Recently, Vardi [Vard90] has announced an algorithm for the linear *boundedness* decision problem for linear sirups, and the uniform boundedness problem can be thought of as a special case of his result. However, it is important to note that the ideas of (non)freeness and uniform boundedness do not exactly coincide, because it is possible for a rule to be simultaneously unbounded and not strongly free:

Example 4.2 The rule

$$r : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \mathtt{e(AX_2)} \ \& \ \mathtt{e(AA)}$$

is unbounded and weakly free, but not strongly free. We can offer only a partial explanation using the tools we have developed so far. First, it is possible to see directly that r is not strongly free, because the derived mapping given by putting $\dot{\phi}(A^{(1)}) = \dot{\phi}(A^{(0)}) = A^{(0)} = A$ suffices to define a containment mapping $\phi : r^2 \to r$. Therefore r has a nontrivial query containment, and is not strongly free. On the other hand, to show that there can be no containment mapping $\phi : r^i \to r^j$ with $i < j$ (that is, to show r is not uniformly bounded, in light of Theorem 4.1), is not so easy. It will suffice to observe that r has an unbounded head chain, and that r has no *cut*, but here we anticipate ourselves. We must wait for our proof of Theorem 4.15, below, to obtain a complete explanation. ∎

In earlier work, [Vard88] and [GMSV87] show (among other things) that linear monadic-recursive and binary-recursive rules can be tested for uniform boundedness, and that boundedness for the latter problem is NP-complete. Abiteboul [Abit89] has shown that boundedness is undecidable for programs with a single recursive rule with nonlinear recursion allowed, but his techniques do not apply either to the study of uniform boundedness, or to the case of *linear* recursive rules.

Our first result in this area is a partial one about "containment depth:"

Theorem 4.3 *Let r be a linear recursive Datalog rule. Then there is a constant K depending only on r such that if the ith recursive expansion r^i contains the jth recursive expansion r^j for some $j > i \geq 1$, then r^i also contains $r^{j'}$ for some j' satisfying $i < j' \leq Ki$.*

We shall prove the theorem in stages through the several intermediate results below.

Perhaps more important than the containment depth result itself is the notation and preparatory lemmas that we shall develop in proving it. Because a substantial gap exists between what boundedness information can be known at a theoretical level and what techniques can be efficiently implemented in a real system, some recent research attention has been given to polynomial-time testable sufficient conditions for boundedness. In the final three sections of the present chapter, we shall take up the study of efficiently testable sufficient conditions for singleton rule set to be unbounded, and we shall arrive at a polynomial-time-testable sufficient condition for unboundedness that simultaneously subsumes several earlier criteria for this problem.

4.1 Notation and preliminaries

We shall first introduce some slightly nonstandard notation for linear recursive rules. After making our formal definitions, we shall pause to relate our notation to our earlier, more familiar representations of linear recursive rules.

Let n be an arbitrary, fixed positive integer. Let $\mathcal{D}_n = \{1, 2, \ldots n\}$, and let $\mathcal{U} = \{A, B, C, \ldots\}$ be an arbitrary finite set of symbols disjoint from $\mathcal{D}_n$. Let $\mathcal{V} = \mathcal{D}_n \cup \mathcal{U}$. A *linear single rule program (sirup) of arity n* is a pair $Q = Q_0 = (h, \mathcal{E})$ where h is a word of length n over the alphabet $\mathcal{V}$, and $\mathcal{E} = \{e_1, \ldots, e_k\}$ is a (possibly empty) finite set of words also over the alphabet $\mathcal{V}$. When a linear sirup Q has at least one word $e_1 \in \mathcal{E}$, we shall also require all such words to have the same fixed length $s = |e_1|$. When speaking of Q,

we shall call h its *recursive subgoal*, the e_i's its *EDB subgoals*, s its *EDB subgoal arity*, $\mathcal{V}$ its *variables*, $\mathcal{D}_n$ its *distinguished variables*, and $\mathcal{U}$ its *nondistinguished variables*. We shall find it convenient to write a linear sirup Q as a string of $k+1$ words over the alphabet $\mathcal{V}$, where we understand the leftmost word to represent h, and where the remaining words (if any) define e_1 through e_k, respectively.

As a first illustration of our system of notation, consider again the familiar transitive closure rule for directed graphs

$$\texttt{path(X}_1\texttt{X}_2\texttt{)} \ :- \ \texttt{path(X}_1\texttt{A) \& edge(AX}_2\texttt{)}.$$

In our system of notation, we would write this rule more economically as

$$1A \quad A2,$$

suppressing the head predicate entirely and all predicate names; also, we have removed the dummy distinguished variable symbol X, replacing $\texttt{X}_1$ and $\texttt{X}_2$ by the symbols 1 and 2, respectively. It is implicit in our representation that the head variables of our rule are distinct symbols, and we shall always make this assumption, as we have in earlier chapters. Because the word $1A$ is written first, we understand it to represent the recursive subgoal h; also implicit in our representation are the facts that $n = |h| = 2$, that $k = 1$, and that $e_1 = A2$ is the only EDB subgoal of Q.

For a second example, consider the rule

$$\texttt{t(XYZ)} \ :- \ \texttt{t(XWZ) \& e(WY) \& e(WZ) \& e(ZZ) \& e(ZY)},$$

which in [Naug86a] is given as an example of a uniformly bounded Datalog rule that nevertheless possesses an unbounded head chain (see Section 4.4). We would write Naughton's rule as

$$1A3 \quad A2 \quad A3 \quad 33 \quad 32,$$

where we have changed the nondistinguished variable W into an A and have substituted the symbols 1, 2, and 3 for the distinguished variables X, Y, and Z, respectively.

In order to refer to particular letters in a word of a linear sirup, we use a *selection operator* σ_i to pick out the ith letter of a word. For example, in our representation of Naughton's linear sirup above, we have $\sigma_3(h) = 3$, $\sigma_1(e_1) = A$, and $\sigma_2(e_4) = 2$. When we

want to refer to the set of all symbols appearing as letters in a particular word, we shall use an unsubscripted σ. Thus we would have $\sigma(h) = \{1, A, 3\}$ and $\sigma(e_3) = \{3\}$ in the example above.

Because our system of notation suppresses all predicate names, it is not powerful enough to directly represent a linear rule such as

$$t(X_1X_2X_3X_4) \ :- \ t(X_1AAX_2) \ \& \ f(X_3BA) \ \& \ g(BA) \ \& \ h(X_4X_2)$$

that has multiple EDB predicate names f, g, and h. However, we can easily prove that deciding uniform boundedness for linear rules with multiple EDB predicates allowed is computationally no harder than deciding the question for linear sirups as we have defined them above, where we have implicitly assumed them to involve at most one (possibly multiply-occurring) EDB predicate.

Lemma 4.4 *There is a polynomial-time algorithm that, given any linear recursive Datalog rule r, returns a linear Datalog rule r' with at most one (possibly multiply-occurring) EDB predicate such that r is uniformly bounded if and only if r' is uniformly bounded.*

Proof Suppose r is a linear recursive Datalog rule with occurrences of different EDB predicate names $\mathrm{f}_1, \ldots, \mathrm{f}_s$. Each predicate name f_i may occur just once, or alternatively several times as the name of an EDB subgoal in r. Suppose the arities of $\mathrm{f}_1, \ldots, \mathrm{f}_s$ are $a_1, \ldots; a_s$, respectively, and suppose that the the arity of the recursive subgoal in r is t. Let $a = 1 + \max(a_i)$. The rule r' will have a recursive subgoal of arity $t + s$ and a single EDB predicate name e of arity a. The EDB subgoals of r' will be in one-to-one correspondence with those of r, and are constructed from them as follows. First, let $X_{\mathrm{f}_1}, \ldots, X_{\mathrm{f}_s}$ be new (distinguished) variable symbols. Suppose f is some EDB subgoal of r, and suppose the predicate name of f is f_i. Then to construct the corresponding EDB subgoal e in r', we let the first a_i variables of e of agree with those of f, and then pad the remaining $a - a_i > 0$ positions with the special distinguished variable X_{f_i}. To construct the recursive subgoal of r' from that of r, we copy the t variables of the recursive subgoal of r into the first t positions of the r' recursive subgoal, and then fill out the remaining s positions with the distinguished variables $X_{\mathrm{f}_1}, \ldots, X_{\mathrm{f}_s}$, respectively. The head of the new rule r', (of course, also of arity $t + s$), is similarly copied from the head of r with the variables $X_{\mathrm{f}_1}, \ldots, X_{\mathrm{f}_s}$ occupying the last s positions.

To verify that the reduction given above is a correct one, it suffices to check that for every containment mapping between two recursive expansions of r there is a containment

mapping between the two corresponding recursive expansions of r', and conversely. But before continuing, we shall first simply give an example of the reduction.

Example 4.5 We return to rule

$$t(X_1X_2X_3X_4) \ :- \ t(X_1AAX_2) \ \& \ f(X_3BA) \ \& \ g(BA) \ \& \ h(X_4X_2),$$

already given above as an example of a rule that has multiple EDB predicate names. Transforming the rule as suggested by Lemma 4.4, we obtain the rule

$$t(X_1X_2X_3X_4X_fX_gX_h) \ :- \ t(X_1AAX_2X_fX_gX_h) \ \& \ e(X_3BAX_f) \ \& \ e(BAX_gX_g) \ \& \ e(X_4X_2X_hX_h),$$

which we claim will be uniformly bounded if and only if the original rule is uniformly bounded. ∎

To see now why the reduction is correct in the general case, we may argue as follows. First, note that by construction of the rule r', the (new) distinguished variables X_{f_i} each have the identity subgoal substitution $(X_{f_i}|X_{f_i})_{r'}$ applied to them when r' is recursively expanded. Therefore every EDB subgoal e of r' with original predicate name f_i in r looks exactly like the corresponding subgoal in the corresponding recursive expansion of r, except that e has a set of padding distinguished variables X_{f_i} occurring in its last $a - a_i$ positions. Because these padding symbols remain unchanged under recursive expansion of r', and because the final (rightmost) padding variable is always in one-to-one correspondence with the original name, X_{f_i} on this subgoal in r, every containment mapping ϕ between recursive expansions of r carries over directly to a corresponding containment mapping ϕ' between corresponding recursive expansions of r', and conversely. Of course, the only difference between these two mappings is that $\phi'(X_{f_i}) = X_{f_i}$ is forced in the latter mapping.

∎

4.2 The substitution graph

Previous authors [Naug86a], [Ioan85], [JAN87] have used various mechanisms to make the process by which a linear recursive Datalog rule is recursively expanded mathematically precise, and in the present thesis we have presented yet another method, which we called "syntactic expansion." However, in the present context we shall prefer the "substitution

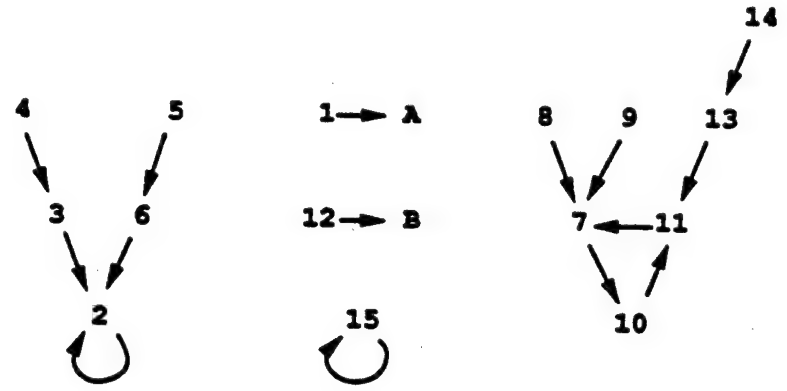
graphs" of [JAN87], although our notation will differ slightly from that paper. Basically, a substitution graph is just a simple way of describing the transformation structure of the subgoal substitutions of Section 2.2.3, making it easier analyze.

More formally, let $\mathcal{Q} = (h, \mathcal{E})$ be a given linear sirup of arity n with a set of k EDB subgoals $\mathcal{E} = \{e_1, \ldots, e_k\}$. The *substitution graph* $S = S(\mathcal{Q})$ has vertices $\mathcal{D}_n \cup \sigma(h)$, and a directed edge $(i, \sigma_i(h))$ for each $i \in \mathcal{D}_n$. In other words, there is an arc from each integer i to the symbol appearing in the ith position of the recursive subgoal of the rule.

For example, consider the following linear sirup $\mathcal{Q}$ of arity 15 (taken from [JAN87]):

$$A22362(10)77(11)7B(11)(13)(15) \quad 1A4B9(12)5(14)8. \tag{4.1}$$

The sirup $\mathcal{Q}$ has one EDB subgoal $e_1 = 1A4B9(12)5(14)8$, and a 15-ary recursive subgoal h. (We have added parentheses to our representations of h and e_1 only in order to allow symbols from D_{15} which exceed 9 to be written unambiguously). The substitution graph $S(\mathcal{Q})$ has vertex set $\mathcal{D}_{15} \cup \{A, B\}$ and the 15 directed edges represented in the following diagram.



We are now almost ready to describe how a general linear sirup $\mathcal{Q} = (h, \mathcal{E})$ is recursively expanded into an infinite sequence of conjunctive queries $\mathcal{Q} = \mathcal{Q}_0, \mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \ldots$ whose union specifies the relation the rule defines. We need a preliminary definition. For every v in $\mathcal{V} = \mathcal{D}_n \cup \mathcal{U}$, we define its *successor symbol* $s(v)$ as follows. First, if v is an element of $\mathcal{D}_n$, then $s(v)$ is just $\sigma_v(h)$; i.e., we simply follow the unique directed edge $(v, \sigma_v(h))$ leading out of v in the substitution graph S to find $s(v)$. Alternatively, we may have $v \in \mathcal{U}$. In this case, we make the preliminary definition that $A = A^0$ for every symbol $A \in \mathcal{U}$, and

then inductively define the successor symbol $s(v^i) = v^{i+1}$ for every integer $i \geq 0$. Thus for example $s(A) = s(A^0) = A^1$, and we have $s(B^{13}) = B^{14}$.

The process by which the rule Q is recursively expanded is now simple to describe. To construct Q_{i+1} from Q_i, we first replace every symbol v occurring in Q_i by its successor symbol $s(v)$. In this way we obtain the new recursive subgoal h^{i+1} for Q_{i-1}, and a subset $\mathcal{E}'$ of its EDB subgoals. Finally, to obtain $\mathcal{E}^{i+1}$, we adjoin to $\mathcal{E}'$ the original EDB subgoals $\mathcal{E}$ of Q, and we are finished.

To illustrate the technique, we return to the transitive closure rule Q

$$1A \quad A2,$$

with substitution graph

We can write the infinite sequence of conjunctive queries $Q = Q_0, Q_1, Q_2, Q_3, \ldots$ as an infinite triangular *query tableau*

$$\begin{array}{lllllll}
Q_0 & 1A^0 & A^02 & & & & \\
Q_1 & 1A^1 & A^1A^0 & A^02 & & & \\
Q_2 & 1A^2 & A^2A^1 & A^1A^0 & A^02 & & \\
Q_3 & 1A^3 & A^3A^2 & A^2A^1 & A^1A^0 & A^02 & \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots
\end{array}$$

where each row is obtained from the previous one by first substituting $s(v)$ for each variable in every word, and then adding the single EDB subgoal $e_1 = A^02$ onto the right-hand end.

For a second example, take Naughton's rule

$$1A3 \quad A2 \quad A3 \quad 33 \quad 32,$$

with substitution graph

The rule has arity $n = 3$ and $k = 4$ EDB subgoals. The query tableau for Naughton's rule begins

$\mathcal{Q}_0$	$1A^03$	A^02	A^03	33	32									
$\mathcal{Q}_1$	$1A^13$	A^1A^0	A^13	33	$3A^0$	A^02	A^03	33	32					
$\mathcal{Q}_2$	$1A^23$	A^2A^1	A^23	33	$3A^1$	A^1A^0	A^13	33	$3A^0$	A^02	A^03	33	32	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Now suppose $\mathcal{Q} = (h, \mathcal{E})$ is an arbitrary linear sirup of arity n with $k \geq 1$ EDB subgoals. (Because a sirup with no EDB subgoals is always uniformly bounded, the case of $k = 0$ will concern us no longer). We need to introduce two more notations in order to refer to particular words inside a query tableau. First, we shall use the (previously introduced) notation h^i to refer to the leftmost word of row i (i.e. the recursive subgoal in the first column of row $\mathcal{Q}_i$). Second, we will use the notation $\mathcal{Q}_i(t, \lambda)$ to refer to the EDB-subgoal word in column $1 + (i - t)k + \lambda$ of row i. When using the latter notation, we shall always be dealing with arguments t and λ that satisfy $0 \leq t \leq i$, and $1 \leq \lambda \leq k$. If either condition fails, $\mathcal{Q}_i(t, \lambda)$ is taken as undefined.

The general ith row $\mathcal{Q}_i$ of the query tableau of a linear sirup $\mathcal{Q}$ with arity n and $k \geq 1$ EDB subgoals therefore consists of $1 + k(i + 1)$ words written left-to-right in the following order:

$$h^i \underbrace{\mathcal{Q}_i(i,1)\cdots\mathcal{Q}_i(i,k)}_{k \text{ words}} \underbrace{\mathcal{Q}_i(i-1,1)\cdots\mathcal{Q}_i(i-1,k)}_{k \text{ words}} \cdots \underbrace{\mathcal{Q}_i(0,1)\cdots\mathcal{Q}_i(0,k)}_{k \text{ words}}.$$

If $q = \mathcal{Q}_i(t, \lambda)$ is such a subgoal word, we shall also occasionally refer to t as the *t-value* of q, and we shall call λ the *λ-value* of q.

Of course, we can also think of the ith-row recursive expansion $\mathcal{Q}_i$ as itself defining a linear sirup $\mathcal{Q}_i = (h^i, \mathcal{E}^i)$, where

$$\mathcal{E}^i = \bigcup_{t,\lambda} \mathcal{Q}_i(t, \lambda).$$

We shall write $\mathcal{Q}^\infty$ for the infinite set $\{\mathcal{Q} = \mathcal{Q}_0, \mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \ldots\}$ of all recursive expansions of $\mathcal{Q}$.

As examples of these notations, in the query tableau corresponding to Naughton's rule we have $h^2 = 1A^23$, $\mathcal{Q}_2(0,2) = A^03$, $\mathcal{Q}_2(2,2) = A^23$, and $\mathcal{Q}_1(1,4) = 3A^0$.

Next, we present some technical lemmas relating to query tableaus and recursive expansion of a given linear sirup $\mathcal{Q}$. Lemma 4.6 is in part merely a verification of the correctness of the recursive expansion procedure as we have outlined it above; still, we shall see that it has some far-reaching consequences in the study of uniform boundedness.

Lemma 4.6 *If $t \leq i$ then $\mathcal{Q}_{i+1}(t,\lambda) = \mathcal{Q}_i(t,\lambda)$.*

Proof We can prove the result by induction on t. When $t = 0$, the result is immediate because $\mathcal{Q}_{i+1}(0,\lambda) = \mathcal{Q}_i(0,\lambda)$ by the definition of the recursive expansion process. Now assume the result to be true for all relevant values of i and λ provided that t is strictly less than some positive integer j, and consider the case of $t = j > 0$. The inductive hypothesis informs us that the two words $\mathcal{Q}_i(j-1,\lambda)$ and $\mathcal{Q}_{i-1}(j-1,\lambda)$ are well-defined and equal; moreover, the recursive expansion process tells us that in order to construct the words $\mathcal{Q}_{i+1}(j,\lambda)$ and $\mathcal{Q}_i(j,\lambda)$, we apply the successor function s to the symbols of $\mathcal{Q}_i(j-1,\lambda)$ and $\mathcal{Q}_{i-1}(j-1,\lambda)$, respectively. Since each successor symbol is defined uniquely, we have $\mathcal{Q}_{i+1}(j,\lambda) = \mathcal{Q}_i(j,\lambda)$, as required, proving the inductive step and completing the proof. ∎

Lemma 4.7 *There is a positive integer p depending only on $\mathcal{Q}$ such that if $i \geq n$ then the following implications are valid:*

1. *If $\sigma_j(h^i)$ is a distinguished variable, then $\sigma_j(h^{i+p}) = \sigma_j(h^i)$.*
2. *If $\sigma_j(h^i) = A^k$ is a nondistinguished variable, then $\sigma_j(h^{i+p}) = A^{k+p}$.*

Lemma 4.7 is the central result of [JAN87] (Theorem 1, page 337), and we shall not reprove it here. The smallest positive integer p satisfying the conditions of Lemma 4.7 is called the *period* of $\mathcal{Q}$. When the substitution graph S of $\mathcal{Q}$ contains at least one cycle, then p is the least common multiple of the cycle lengths in S. If the substitution graph of $\mathcal{Q}$ contains no cycles, then taking $p = 1$ will satisfy the conditions of the lemma.

Lemma 4.8, below, is also implicit in the results of [JAN87], and we shall omit a formal proof of this result as well. Intuitively, the result expresses the idea that the variable

patterns of the EDB subgoals in recursive expansions of a linear sirup $\mathcal{Q}$ of arity n also follow easily described patterns after the nth recursive expansion $\mathcal{Q}_n$.

Lemma 4.8 *Let $\mathcal{Q}$ be a linear sirup of arity n, and suppose the period of $\mathcal{Q}$ is p. If $i \geq n$ and $t \geq n$, then the following implications are valid.*

1. *If $\sigma_j(\mathcal{Q}_i(t,\lambda))$ is a distinguished variable, then $\sigma_j(\mathcal{Q}_{i+p}(t+p,\lambda)) = \sigma_j(\mathcal{Q}_i(t,\lambda))$.*
2. *If $\sigma_j(\mathcal{Q}_i(t,\lambda)) = A^k$ is a nondistinguished variable, then $\sigma_j(\mathcal{Q}_{i+p}(t+p,\lambda)) = A^{k-p}$.*

To study containment mappings ϕ between the various recursive expansions $\mathcal{Q}_i$, we need to translate our earlier description of these mappings (conditions 1–4 from Section 2.2.1, above) into the present system of notation.

Let $\mathcal{Q} = (h, \mathcal{E})$ be a linear sirup of arity n with k EDB subgoals. Then a function $\phi : \mathcal{Q}_i \to \mathcal{Q}_j$ will be a containment mapping if it satisfies the following properties:

1. $\phi(h^i) = h^j$.
2. $\phi(\mathcal{Q}_i(t,\lambda)) \in \mathcal{E}^j$ for all t and λ.
3. If $\sigma_l(h^i)$ is a distinguished variable, then $\sigma_l(\phi(h^i)) = \sigma_l(h^j)$ is the same distinguished variable.
4. If $\sigma_l(\mathcal{Q}_i(t,\lambda))$ is a distinguished variable, then $\sigma_l(\phi(\mathcal{Q}_i(t,\lambda)))$ is the same distinguished variable.
5. If $\sigma_{l_1}(\mathcal{Q}_i(t_1,\lambda_1)) = \sigma_{l_2}(\mathcal{Q}_i(t_2,\lambda_2))$ then $\sigma_{l_1}(\phi(\mathcal{Q}_i(t_1,\lambda_1))) = \sigma_{l_2}(\phi(\mathcal{Q}_i(t_2,\lambda_2)))$.
6. If $\sigma_{l_1}(\mathcal{Q}_i(t_1,\lambda_1)) = \sigma_{l_2}(h^i)$ then $\sigma_{l_1}(\phi(\mathcal{Q}_i(t_1,\lambda_1))) = \sigma_{l_2}(\phi(h^i))$.

Here, we can see that Conditions 1 and 2 above are just restatements of our earlier Conditions 1 and 2 from Section 2.2.1, while the present Conditions 3 and 4 jointly restate Condition 3 from Section 2.2.1. Finally, Conditions 5 and 6 above restate Condition 4 from Section 2.2.1.

A containment mapping $\phi : \mathcal{Q}_i \to \mathcal{Q}_j$ is called *nontrivial* if $i \neq j$. We have already seen that the relation $\succeq$ on $\mathcal{Q}^\infty$ given by $\mathcal{Q}_i \succeq \mathcal{Q}_j$ if and only if there is a containment mapping $\phi : \mathcal{Q}_i \to \mathcal{Q}_j$ is reflexive and transitive.

Recall that $\mathcal{Q}$ is called *uniformly bounded* if there is a constant N such that for all $j \geq N$, we have $\mathcal{Q}_i \succeq \mathcal{Q}_j$ for some $i < N$.

Also, recall that corresponding to every containment mapping $\phi : \mathcal{Q}_i \rightarrow \mathcal{Q}_j$ there is a unique *derived mapping*

$$\dot{\phi} : \sigma(\mathcal{Q}_i) \rightarrow \sigma(\mathcal{Q}_j)$$

which is a mapping between the variables occurring in $\mathcal{Q}_i$ and those occurring in $\mathcal{Q}_j$ and is given by the two rules that

$$\dot{\phi}(\sigma_l(h^i)) = \sigma_l(h^j) \text{ for every } l$$

and

$$\phi(\mathcal{Q}_i(t_1, \lambda_1)) = \mathcal{Q}_j(t_2, \lambda_2) \Longrightarrow \dot{\phi}(\sigma_l(\mathcal{Q}_i(t_1, \lambda_1))) = \sigma_l(\mathcal{Q}_j(t_2, \lambda_2)) \text{ for every } l.$$

That the mapping $\dot{\phi}$ is well-defined by these two rules is a simple consequence of the axioms for a containment mapping.

There are several technical results which we shall need to verify before continuing:

Lemma 4.9 *Let $\mathcal{Q}$ be a linear sirup of arity n. If A^l is a nondistinguished variable occurring in $\mathcal{Q}_i(t, \lambda)$, then $0 \leq t - l \leq n$.*

Proof We prove the result by induction on i. First, note that if $i = 0$, then $t = l = 0$ and the result is trivially true. Inductively, suppose that the lemma is valid for all values i strictly less than some fixed integer M, and consider the case $i = M$. First, note that if $l > 0$, then i and t are both ≥ 1 and the EDB subgoal $\mathcal{Q}_{i-1}(t-1, \lambda)$ must contain the nondistinguished variable A^{l-1} in order for $\mathcal{Q}_i(t, \lambda)$ to contain A^l. The induction hypothesis then implies $0 \leq (t-1) - (l-1) \leq n$; in other words, $0 \leq t - l \leq n$, and we are finished.

Alternatively, we may have $l = 0$. Suppose that $\sigma_j(\mathcal{Q}_i(t, \lambda)) = A^l = A^0$. In this case, if it is not already the case that $t = 0$, then by the definition of the recursive expansion process, there are exactly t different nonpersistent variables $\sigma_j(\mathcal{Q}_{i-1}(t-1, \lambda)), \sigma_j(\mathcal{Q}_{i-2}(t-2, \lambda)), \ldots, \sigma_j(\mathcal{Q}_{i-t}(0, \lambda))$ "above" $\sigma_j(\mathcal{Q}_i(t, \lambda))$ in the query tableau for $\mathcal{Q}$. Since n is a upper bound on the number cf such nonpersistent variables, we have $0 \leq t \leq n$, or in other words, $0 \leq t - l \leq n$, as required. ∎

A simple corollary of the previous Lemma is

Lemma 4.10 *Suppose $A^{l_1} \in \sigma(\mathcal{Q}_i(t_1, \lambda_1))$ and $B^{l_2} \in \sigma(\mathcal{Q}_i(t_2, \lambda_2))$. Then $|(t_1 - t_2) - (l_1 - l_2)| \leq n$.*

Proof Applying Lemma 4.9, we find that $0 \leq t_1 - l_1 \leq n$ and $0 \leq t_2 - l_2 \leq n$. Subtracting, we conclude that $|(t_1 - t_2) - (l_1 - l_2)| \leq n$, as required. ∎

A second useful corollary of Lemma 4.9 is

Lemma 4.11 *Suppose* $\sigma(\mathcal{Q}_i(t_1, \lambda_1)) \cap \sigma(\mathcal{Q}_i(t_2, \lambda_2))$ *contains some nondistinguished variable* A^l. *Then* $|t_1 - t_2| \leq n$.

Proof We apply the previous lemma with $A = B$ and $l_1 = l_2 = l$. ∎

Finally, it is useful to have a characterization of the exponents that can appear on the nondistinguished variables in the recursive subgoal h^i of a recursive expansion $\mathcal{Q}_i$:

Lemma 4.12 *Suppose* A^l *occurs in* h^i, *the recursive subgoal of the* i*th recursive expansion* $\mathcal{Q}_i$ *of a linear sirup* $\mathcal{Q}$ *of arity* n. *Then* $\max(0, i - n) \leq l \leq i$.

Proof We may prove the lemma just as we proved Lemma 4.9, above, by induction on i. First, note that if $i = 0$, then $l = 0$ and the lemma is trivially true. Inductively, suppose that the lemma is valid for all values i strictly less than some fixed integer M, and consider the case $i = M$. If $l > 0$, then $i \geq 1$ and we see that h^{i-1} must contain the nondistinguished variable A^{l-1} in order for h^i to contain A^l. The induction hypothesis then implies that $\max(0, i - 1 - n) \leq l - 1 \leq i - 1$, which implies $\max(0, i - n) \leq l \leq i$, as required.

Alternatively, we may have $l = 0$. Suppose that $\sigma_j(h^i) = A^l = A^0$. In this case, if it is not already the case that $i = 0$, then by the definition of the recursive expansion process, there are exactly i different nonpersistent variables $\sigma_j(h^{i-1}), \sigma_j(h^{i-2}), \ldots, \sigma_j(h^0)$ "above" $\sigma_j(h^i)$ in the query tableau for $\mathcal{Q}$. Since n is a upper bound on the number of such nonpersistent variables, we have $i \leq n$, or in other words, because $l = 0$, we have $\max(0, i - n) \leq l \leq i$, as required. ∎

4.3 Containment depth

At last we are in a position to be able to prove the result already alluded to above:

Theorem 4.13 *There is a constant* K *depending only on* $\mathcal{Q}$ *such that if* $\mathcal{Q}_i \succeq \mathcal{Q}_j$ *for some* $j > i \geq 1$, *then* $\mathcal{Q}_i \succeq \mathcal{Q}_{j'}$ *for some* j' *satisfying* $i < j' \leq Ki$.

Proof We shall see that the constant K may be taken as roughly proportional to the product pnk of the period p of $\mathcal{Q}$, its recursive subgoal arity n and the number k of its EDB subgoals; more precisely, we claim that the choice $K = 4pnk$ suffices. Our method is essentially a "reverse pumping argument" as has been used for example in [Cosm88].

Fix i, and consider the smallest $j' > i$ such that $\mathcal{Q}_i \succeq \mathcal{Q}_{j'}$. To prove the theorem, we shall show that if $\mathcal{Q}_i \succeq \mathcal{Q}_j$ and $j > Ki$, then $j \neq j'$; i.e., j is not minimal.

So suppose that $\mathcal{Q}_i \succeq \mathcal{Q}_j$, and $j > Ki$. Let $\phi : \mathcal{Q}_i \to \mathcal{Q}_j$ be a containment mapping. $\mathcal{Q}_j$ is represented by the jth row of the query tableau for $\mathcal{Q}$. This row is

$$h^j \underbrace{\mathcal{Q}_j(i,1)\cdots\mathcal{Q}_j(j,k)}_{k \text{ words}}\underbrace{\mathcal{Q}_j(j-1,1)\cdots\mathcal{Q}_j(j-1,k)}_{k \text{ words}}\cdots\underbrace{\mathcal{Q}_j(0,1)\cdots\mathcal{Q}_j(0,k)}_{k \text{ words}},$$

and it consists of the recursive subgoal h^j and $k(j+1)$ EDB subgoals. We shall focus on the first $(j-n)k$ EDB subgoals from the left in $\mathcal{Q}_j$, and shall think of the leftmost of these subgoals as broken into $T = \lfloor (j-n)/(pn) \rfloor$ "blocks" of pnk EDB subgoals, apiece. Note that

$$j - n > Ki - n = 4pnki - n = (4pk-1)n \geq 3np,$$

so we are definitely looking at a nonempty collection of subgoals here. Now $\mathcal{Q}_i$ has only $k(i+1)$ EDB subgoals, and because

$$\begin{aligned} j &> Ki \\ &= 4pnki \\ &> n + pnk(i+1), \end{aligned}$$

we conclude that

$$T > (i+1)k.$$

By the pigeonhole principle, there is some block of $\mathcal{Q}_j$ into which ϕ maps no EDB subgoal $\mathcal{Q}_i(t,\lambda)$. Let

$$\underbrace{\mathcal{Q}_j(M,1)\cdots\mathcal{Q}_j(M,k)}_{k \text{ words}}\cdots\underbrace{\mathcal{Q}_j(M-pn+1,1)\cdots\mathcal{Q}_j(M-pn+1,k)}_{k \text{ words}}$$

be the first (i.e., leftmost: choose the maximal such M) block of $\mathcal{Q}_j$ into which ϕ maps no EDB subgoal of $\mathcal{Q}_i$. Note that the value M falls in the closed interval $[j - \lfloor (j-n)/(pn) \rfloor + 1, j]$, and in particular we have

$$M \geq j - \lfloor \frac{j-n}{pn} \rfloor + 1 \geq j - \frac{j-n}{pn} + 1.$$

We claim that in fact $M \geq n$, also. To see this, we can write

$$\begin{aligned} pn^2 &\leq pn^2 - n - n(p-1) \\ &\leq (pn-1)j + n(p-1) \\ &= pnj - (j-n) + pn, \end{aligned}$$

and dividing both sides of this inequality by pn we obtain

$$n \leq j - \frac{j-n}{pn} + 1 \leq M,$$

as claimed.

To finish the proof, we shall show that there is a second containment mapping

$$\phi_1 : Q_i \rightarrow Q_{j-p}$$

that is constructible from ϕ and M as follows. First, we define $\phi_1(h^i) = h^{j-p}$, as we must. Then, to define $\phi_1(Q_i(t_1, \lambda_1))$ for an EDB subgoal of Q_i, we first examine the image $\phi(Q_i(t_1, \lambda_1)) = Q_j(t_2, \lambda_2)$. By our arguments above, we must have either $t_2 > M$ or $t_2 < M - pn + 1$: if the former is the case, then we define $\phi_1(Q_i(t_1, \lambda_1)) = Q_{j-p}(t_2 - p, \lambda_2)$: and if the latter, then we define $\phi_1(Q_i(t_1, \lambda_1)) = Q_{j-p}(t_2, \lambda_2)$.

Next, we check that the t-values for Q_{j-p} we have just defined lie in the proper ranges.

Consider first the case of $t_2 > M$. We need to check that $0 \leq t_2 - p \leq j - p$. We easily see that $t_2 - p \leq j - p$ because $t_2 \leq j$, by assumption. To verify that $0 \leq t_2 - p$, it is enough to check that $p \leq M$, which we can do as follows. We know that the first block of Q_j not mapped into by ϕ can be the $(1 + (i+1)k)$th block from the left, at the latest. Therefore $M \geq j - (i+1)k$; we need to check that $j - (i+1)k \geq p$ to finish. Now

$$\begin{aligned} j - (i+1)k &\geq 4pnki - (i+1)k \\ &\geq 2ki(pn) \\ &> p, \end{aligned}$$

as required.

Now consider the second case of the definition above, i.e. the case of $t_2 < M - pn + 1$. We need to check in this case that $0 \leq t_2 \leq j - p$. We already know that $0 \leq t_2$ by assumption, so the problem reduces to showing that $t_2 \leq j - p$. To see this, note that if $t_2 < M - pn + 1$, then there are at least pn blocks to the left of the t_2 block in Q_j, none

of which is mapped into by ϕ. Therefore $j \geq t_2 - pn \geq t_2 - p$, so we have $t_2 \leq j - p$, as required.

All that remains to done is to check that ϕ_1 is indeed a containment mapping; we must verify the six points of the definition in turn. Because these points are basically just mechanical verifications, we shall place them in the appendix for the sake of clarity.

∎

If additionally one could give an explicit bound in terms of r on the maximal depth i at which the first nontrivial containment $r^i \succeq r^j$ occurs for a bounded rule r in the previous theorem, then we would easily obtain an alternative proof of Vardi's result that the boundedness problem is decidable, but such a proof has thus far eluded us.

4.4 Cuts

Let $Q = (h, \mathcal{E})$ be a linear sirup of arity n with k EDB subgoals, and let $Q_i = (h^i, \mathcal{E}^i)$ be a recursive expansion of Q. The *linking graph* $\mathcal{G}_i$ is an undirected graph on the vertices $h^i \cup \mathcal{E}^i$, with an edge between two words if they share a nondistinguished variable. The connected component of $\mathcal{G}_i$ to which h^i belongs is called the *head chain* of Q_i. If the number of vertices belonging to the head chain of Q_i is unbounded as i increases, then we say that Q has an *unbounded head chain*. The following result is from [Naug86a], although we have slightly changed the terminology used in that paper:

Lemma 4.14 *Let Q be a linear single rule program. If Q is not uniformly bounded, then Q has an unbounded head chain.*

Naughton and Sagiv [Naug86a], [NaSa87] and Ioannidis [Ioan85] give efficient (linear-time) algorithms for deciding whether a linear sirup Q has an unbounded head chain. Unfortunately, a rule may have an unbounded head chain and still be uniformly bounded. Naughton's rule,

$$1A3 \quad A2 \quad A3 \quad 33 \quad 32,$$

already mentioned above, is perhaps the simplest example of such a rule. These results led Naughton and Sagiv [NaSa87] to study sufficient conditions for a rule with an unbounded head chain to be in fact unbounded. They present four different polynomial-time testable

conditions, each sufficient for a rule with an unbounded head chain rule to be unbounded. Here, we generalize these results to give a single polynomial-time testable sufficient condition for the unboundedness of a rule with an unbounded head chain that simultaneously subsumes all four classes identified by Naughton and Sagiv. We shall call our condition the *cut condition*. Along the way, we shall need to prove two preparatory lemmas that closely parallel lemmas due to Naughton and Sagiv [NaSa87]—in particular, our Lemmas 4.16 and 4.17 correspond to Naughton and Sagiv's Lemmas 4.2 and 4.5, respectively. Our alternative proofs of these two lemmas, however, together with our more detailed notation, will allow us to deduce some additional information that will lead to the cut condition.

First, we must continue with some more definitions. We shall call a distinguished variable $d \in \mathcal{D}_n$ of Q *persistent* if d belongs to some (possibly trivial) directed cycle in the substitution graph S of Q. To state it another way, d is persistent if by following successor symbols away from d in S we find that $s^t(d) = d$ for some $t > 0$. Otherwise, d will be called *nonpersistent*. (The two definitions follow those of [Naug86a]). If a distinguished variable d is nonpersistent, then there are two possible subcases: either there is a unique positive integer $t \leq n$ and a unique nondistinguished variable $A \in \mathcal{U}$ such that $s^t(d) = A = A^0$, in which case we shall say that the variable d *belongs to* A, and that d has *height* t; or alternatively, the nonpersistent variable d must belong to a component of the substitution graph S of Q that contains a cycle (but d is not actually in the cycle), in which case we shall say that d is a *stem variable*.

For example, in the rule 4.1, above, the distinguished variables 2, 7, 10, 11, and 15 are persistent, while the remaining distinguished variables 1, 3, 4, 5, 6, 8, 9, 12, 13, and 14 are nonpersistent. Amongst the nonpersistent variables, 1 and 12 belong to A and B respectively, while 3, 4, 5, 6, 8, 9, 13, and 14 are stem. The heights of the variables 1 and 12 are both equal to one.

If a nonpersistent variable d belongs to A and the height t of d is maximal amongst all nonpersistent variables of Q belonging to A, then we call d a *leader*, or alternatively an *A-leader* when we wish to remind ourselves to which nondistinguished variable the variable d belongs. We shall also say that the variable A itself belongs to A, and we define the height t of A to be zero.

Next, let e_i be an EDB subgoal of the linear sirup Q, and let d be some fixed distinguished variable of Q. Then we shall say that e_i is *A-leader-containing* if $\sigma_s(e_i)$ is an A-leader for some s. Finally, we shall say that a second EDB subgoal e_j is a (d, A)-*cut* of e_i if the

following properties hold:

1. e_i is A-leader-containing.
2. d is either a stem variable or a persistent variable.
3. In some position s of e_i in which an A-leader appears, e_j has the variable d.
4. If $\sigma_s(e_i)$ is persistent or stem, then $\sigma_s(e_i)$ and $\sigma_s(e_j)$ are in the same component of the substitution graph of $\mathcal{Q}$ (so in particular $\sigma_s(e_j)$ is either persistent or stem).
5. If $\sigma_s(e_i)$ belongs to some nondistinguished variable B but is not a B-leader, then $\sigma_s(e_j)$ belongs to some nondistinguished variable C (we allow $A = B$ and/or $B = C$ here).
6. If $\sigma_s(e_i)$ is an A-leader, then $\sigma_s(e_j)$ belongs to the same component as d in the substitution graph of $\mathcal{Q}$.

Notice, in particular, that no restrictions are placed on leaders that belong to nondistinguished variables other than A. We say more simply that $\mathcal{Q}$ has a *cut* if e_j is a (d, A)-cut of e_i for some appropriate values d, A, i, and j.

Our main goal in the present section is to prove:

Theorem 4.15 *If $\mathcal{Q}$ is uniformly bounded and has an unbounded head chain, then $\mathcal{Q}$ has a cut.*

We shall require two intermediate results (Lemmas 4.16 and 4.17) before we prove the theorem. Before we continue, it is useful to reexamine Naughton's rule once more in order to get some intuition for the eventual proof of Theorem 4.15, below.

In Naughton's rule, the EDB subgoal $e_2 = A3$ is a $(3, A)$-cut of $e_1 = A2$. We can think of this cut as arising from a containment mapping in the following way. We found above that the query tableau for Naughton's rule begins:

$\mathcal{Q}_0$	$1A^03$	A^02	A^03	33	32									
$\mathcal{Q}_1$	$1A^13$	A^1A^0	A^13	33	$3A^0$	A^02	A^03	33	32					
$\mathcal{Q}_2$	$1A^23$	A^2A^1	A^23	33	$3A^1$	A^1A^0	A^13	33	$3A^0$	A^02	A^03	33	32	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

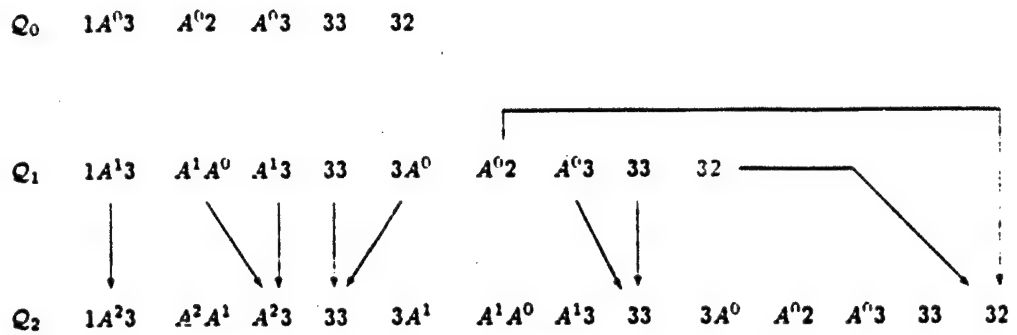


Figure 4.1: A containment mapping $\phi : Q_1 \rightarrow Q_2$ in Naughton's rule.

Exhaustively checking possible mappings, one finds that there is no containment mapping $\phi : Q_0 \rightarrow Q_1$, and also that there is no containment mapping $\phi : Q_0 \rightarrow Q_2$. However, there is a containment mapping $\phi : Q_1 \rightarrow Q_2$, as indicated by the arrows in Figure 4.1.

Several points about the mapping ϕ can be made:

1. There is a nondistinguished variable of Q_1 (here, the variable A^0) that ϕ maps to a persistent distinguished variable (here, the variable 3).

2. The variable A^0 is on the head chain of Q_1.

3. Looking straight above the leftmost occurrence of A^0 in Q_1 in the top row of the query tableau, we encounter the EDB subgoal $e_1 = A2$, which contains the A-leader variable 2.

4. Looking above the *image* of the leftmost A^0-containing subgoal in Q_1, (in other words, looking above the subgoal A^23 in Q_2), we encounter the EDB subgoal $e_2 = A3$, which is a $(3, A)$-cut of e_1.

Basically, the idea of the cut condition is this: the above derivation of a cut in Q_0 from a containment mapping $\phi : Q_1 \rightarrow Q_2$ can be generalized. In fact, we shall see that whenever a containment mapping $\phi : Q_i \rightarrow Q_j$ (with $i < j$) exists in a query tableau, then we can produce a cut in the rule Q_0 itself by "looking above" certain subgoals in Q_i and Q_j.

We return now to the preparatory lemmas mentioned above.

Our first preparatory Lemma asserts that when $\mathcal{Q}$ has an unbounded head chain, there are EDB subgoals that contain nonpersistent variables and belong to the head chain of $\mathcal{Q}_i$ in every recursive expansion $\mathcal{Q}_i$:

Lemma 4.16 *Suppose $\mathcal{Q}$ is a linear sirup of arity n with k EDB subgoals, and suppose $\mathcal{Q}$ has an unbounded head chain. Let $\mathcal{Q}_i$ be an arbitrary recursive expansion of $\mathcal{Q}$. Then there is an EDB subgoal $e = \mathcal{Q}_i(t, \lambda)$ of $\mathcal{Q}_i$ such that the following conditions are met:*

1. *e belongs to the head chain of $\mathcal{Q}_i$.*
2. *e contains a nonpersistent variable.*

Proof Suppose to the contrary that $\mathcal{Q}_i$ is a recursive expansion of $\mathcal{Q}$ that has no nonpersistent-variable-containing EDB subgoals on its head chain. First, note that the head chain of every $\mathcal{Q}_l$ with $l > i$ contains at least as many EDB subgoals as the head chain of $\mathcal{Q}_i$, because if $\mathcal{Q}_i(t, \lambda)$ belongs to the head chain of $\mathcal{Q}_i$, then we easily see that $\mathcal{Q}_{i+a}(t+a, \lambda)$ belongs to the head chain of $\mathcal{Q}_{i+a}$. Moreover, because such a $\mathcal{Q}_i(t, \lambda)$ contains no nonpersistent variables, neither does $\mathcal{Q}_{i+a}(t + a, \lambda)$. Now because $\mathcal{Q}$ has an unbounded head chain, there is some $j > i$ such that the recursive expansion $\mathcal{Q}_j$ has strictly more EDB subgoals on its head chain than $\mathcal{Q}_i$ has on its head chain. Choosing the minimal such j, we claim that $\mathcal{Q}_{j-1}$ must have a nonpersistent variable on its head chain, a contradiction.

To prove the claim, let $q_1 = \mathcal{Q}_j(t_1, \lambda_1)$ be a "new" member of the head chain of $\mathcal{Q}_j$ (i.e. $\mathcal{Q}_j(t_1, \lambda_1)$ is not a shift $\mathcal{Q}_{i+a}(t + a, \lambda)$ of some subgoal $\mathcal{Q}_i(t, \lambda)$ on the head chain of $\mathcal{Q}_i$, with $j = i + a$). We may also assume that q_1 is chosen so that it shares a nondistinguished variable A^l with some $q_2 = \mathcal{Q}_{i+a}(t_2 + a, \lambda_2)$, such that $i + a = j$ and $\mathcal{Q}_i(t_2, \lambda_2)$ is on the head chain of $\mathcal{Q}_i$. First, note that l must be zero, for if $l > 0$, then $q_1' = \mathcal{Q}_{j-1}(t_1 - 1, \lambda_1)$ is similarly a new member of the head chain of $\mathcal{Q}_{j-1}$, and we have chosen j minimal. So $l = 0$ and we see that $\mathcal{Q}_{i+a-1}(t_2 + a - 1, \lambda_2)$ must contain a nonpersistent variable in order for this A^0 to arise in q_2. Because this $\mathcal{Q}_{i+a-1}(t_2 + a - 1, \lambda_2)$ is on the head chain of $\mathcal{Q}_{i+a-1}$, we have our contradiction. ∎

Next, we prove that if a rule $\mathcal{Q}$ with an unbounded head chain is bounded, then there must be a containment mapping $\phi : \mathcal{Q}_i \rightarrow \mathcal{Q}_j$ between two recursive expansions of $\mathcal{Q}$ with $i < j$ such that the derived mapping $\dot{\phi}$ maps some nondistinguished variable A^l to a persistent distinguished variable d:

Lemma 4.17 *Suppose Q is a linear sirup of arity n with k EDB subgoals, suppose Q has an unbounded head chain, and suppose that Q is bounded. Then there are integers i, j, and l, and a persistent distinguished variable d such the following conditions are met:*

1. *There is a containment mapping $\phi : Q_i \rightarrow Q_j$ with $i < j$.*
2. *$\phi(A^l) = d$ for some nondistinguished variable A^l of Q_i.*
3. *A^l occurs in an EDB predicate on the head chain of Q_i.*

Proof Because Q is bounded, there is a containment mapping $\phi' : Q_i \rightarrow Q_{j'}$ for some two values $i < j'$ (Theorem 4.1). Also, by applying the splicing lemma repeatedly if necessary, we can assume that $i \geq n$ without any loss of generality. By Lemma 4.15, we know that the head chain of Q_i has some nonpersistent-variable-containing subgoal $q = Q_i(t, \lambda)$ belonging to it. Let $h^i = q_0, q_1, q_2, \ldots, q_P = q$ be a shortest sequence of subgoals of Q_i that "connects" q to the head chain of Q_i; in other words, for each pair (q_i, q_{i-1}) we have that q_i and q_{i+1} share some nondistinguished variable, and P is chosen as minimal with respect to sequences that connect h^i to q. (Such a sequence is most easily thought of as a shortest path between h^i and q in the linking graph G_i of Q_i). Because Q_i has exactly $k(i+1)$ EDB subgoals, we have $P \leq k(i+1)$ here.

Now suppose that contrary to the present theorem statement, there is no containment mapping ϕ from one recursive expansion of Q to another, deeper one that maps a nondistinguished variable to a persistent variable d. We would like to obtain a contradiction. In our proof of Theorem 4.1, we saw that $Q_i \succeq Q_{j'}$ implies that $Q_i \succeq Q_{mj'-(m-1)i}$ for every positive integer m. In particular, consider the choice $m = 4nk(i+1)$, let $j = mj' - (m-1)i$, and suppose that $\phi : Q_i \rightarrow Q_j$ is an associated containment mapping. We shall show that our combined hypotheses imply that $\phi(q)$ cannot contain a nonpersistent variable (as it must, if ϕ is to be a containment mapping)—a contradiction.

In fact, we shall need to make the slightly stronger claim that the t-value of $\phi(q_i)$ exceeds $(P + 2 - i)n$ for every i in the range $1 \leq i \leq P$. Once we have proved the claim, we shall know in particular that the t-value of $\phi(q_P) = \phi(q)$ is greater than $2n$. Because no EDB subgoal with a t value greater than n can have a nonpersistent variable occurring in it, we shall have our contradiction.

We can prove the claim by induction on i. First, because $h^i = q_0$ and q_1 share a nondistinguished variable, and $\phi(h^i) = h^j$ has all of its nondistinguished variable's exponents in

the range $[j-n, n]$ (Lemma 4.12), we see that the t-value t of $\phi(q_1)$ in $\mathcal{Q}_j$ can be no smaller than $j-n$, by Lemma 4.9. Now

$$\begin{aligned} j-n &= mj' - (m-1)i - n \\ &= m(j'-i) + (i-n) \\ &\geq 4nk(i+1)(j'-i) \\ &\geq (P+1)n, \end{aligned}$$

so $t \geq (P+1)n$, as required in the base case of our claim.

Inductively, suppose the claim is valid for a value $i = M-1 < P$ and consider the case of $i+1 = M$. We know that q_i and q_{i+1} share some nondistinguished variable A^l. The induction hypothesis implies that $\phi(q_i)$ has a t-value $\geq (P+2-i)n \geq n$, so $\dot{\phi}(A^l)$ cannot be a nonpersistent distinguished variable, and by assumption $\dot{\phi}(A^l)$ is not a persistent distinguished variable. Therefore $\dot{\phi}(A^l) = B^s$ for some nondistinguished variable B with exponent s, say; in other words, $\phi(q_i)$ and $\phi(q_{i+1})$ share a nondistinguished variable. Applying Lemma 4.11, we find that the condition that the t-value of $\phi(q_i)$ exceeds $(P+2-i)n$ implies that the t-value of $\phi(q_{i+1})$ can be no smaller than $(P+2-i)n-n = (P+2-(i+1))n$, as we were required to show. This completes the proof of the claim, and of the lemma.

∎

We can return now to the proof of Theorem 4.15:

Proof Let $\mathcal{Q}$ be a linear sirup of arity n with k EDB subgoals, and suppose $\mathcal{Q}$ has an unbounded head chain, yet $\mathcal{Q}$ is bounded. We need to show $\mathcal{Q}$ has a cut. By Lemma 4.17, there are integer tuples (i, j, l, d') such that $i < j$, and we have a containment mapping $\phi : \mathcal{Q}_i \rightarrow \mathcal{Q}_j$, where $\dot{\phi}(A^l) = d'$ for some nondistinguished variable A of exponent l in $\mathcal{Q}_i$, and d' is some persistent distinguished variable. By applying the splicing lemma and the transitive property of containment repeatedly if necessary, we can assume without any loss of generality that the tuple (i, j, l, d') satisfies the conditions $i \geq 3n$ and $j \geq 4nk(i+1)$, and that $\phi(\mathcal{Q}_i(i-\alpha, \lambda)) = \phi(\mathcal{Q}_j(j-\alpha, \lambda))$ for all α in the range $0 \leq \alpha \leq n$. Now suppose that $q = \mathcal{Q}_i(t_1, \lambda_1)$ is the leftmost EDB subgoal on the head chain of $\mathcal{Q}_i$ that contains a variable A^l such that $\dot{\phi}(A^l) = d'$ for some persistent variable d', and suppose that $\phi(q) = \phi(\mathcal{Q}_i(t_1, \lambda_1)) = \mathcal{Q}_j(t_2, \lambda_2)$, and $\sigma_v(q) = A^l$. It follows that $t_1 \leq i-n$ here.

We claim that the EDB subgoal e_{λ_2} is a (d, A)-cut of e_{λ_1} in $\mathcal{Q}$, where $d = \sigma_v(e_{\lambda_2})$ defines d.

To prove the claim, we must verify the six points 1–6 of the definition of a cut, above.

To verify condition 1 for a cut, suppose that $\sigma_v(q) = A^l$, as above. Then we claim that $\sigma_v(e_{\lambda_1})$ is an A-leader. Clearly $\sigma_v(e_{\lambda_1})$ belongs to A, by the definition of the recursive expansion process. We must show that $\sigma_v(e_{\lambda_1})$ has maximal height amongst variables of $\mathcal{Q}$ belonging to A. Every A-leader must occur in some EDB subgoal of $\mathcal{Q}$, for otherwise $\mathcal{Q}$ is not safe. Suppose then that e_λ is an EDB subgoal of $\mathcal{Q}$ containing an A-leader $\sigma_k(e_\lambda)$, and that $\sigma_v(e_{\lambda_1})$ has a strictly smaller height than $\sigma_k(e_\lambda)$. Writing

$$\text{height}(\sigma_k(e_\lambda)) = \beta + \text{height}(\sigma_v(e_{\lambda_1})),$$

for some β with $n \geq \beta > 0$, we see that

$$\sigma_k(\mathcal{Q}_i(t_1 + \beta, \lambda)) = A^l = \sigma_v(\mathcal{Q}_i(t_1, \lambda_1)).$$

However, because $\mathcal{Q}_i(t_1 + \beta, \lambda)$ occurs to the left of q in the query tableau representation of $\mathcal{Q}_i$, we know that $\dot{\phi}(A^l)$ can't be distinguished, by our choice of q—a contradiction.

To verify condition 2 for a cut, it suffices to note that because d' is a persistent variable which arises in $\phi(q)$ by following successor symbols away from the symbol d in e_{λ_2}, the variable d must be a stem or persistent variable in the same component as d' of the substitution graph of $\mathcal{Q}$.

The verification of condition 3 for a cut is immediate by our construction, because we have defined d as equal to $\sigma_v(e_{\lambda_2})$, where e_{λ_1} has an A-leader at position v.

To verify condition 4 for a cut, suppose $\sigma_s(e_{\lambda_1}) = d_0$ is persistent or stem. Then by the definition of the recursive expansion process, q has a persistent variable d_1 at position s, and d_0 and d_1 must belong to the same component of the substitution graph of $\mathcal{Q}$. Also, because ϕ is a containment mapping, we have $\sigma_s(\phi(q)) = d_1$. Since d_1 can appear at position s in $\phi(q)$ only if e_{λ_2} has a variable from the same component as d_1 at position s, we see that $\sigma_s(e_{\lambda_1})$ and $\sigma_s(e_{\lambda_2})$ must belong to the same component of the substitution graph of $\mathcal{Q}$, as required.

To verify condition 5 for a cut, suppose $\sigma_s(e_{\lambda_1})$ belongs to a nondistinguished variable B, but is not a B-leader. Now suppose that contrary to the statement of condition 5, $\sigma_s(e_{\lambda_2})$ does not belong to some nondistinguished variable C, but is on the contrary either a stem or a persistent variable. Then $\sigma_s(\phi(q))$ is some persistent variable d_0, and $\dot{\phi}$ carries some B^{l_0} to d_0. But because $\sigma_s(e_{\lambda_1})$ does not have maximal height, we now have a contradiction as in our verification of condition 1 for a cut, above, by our choice of q.

Finally, to verify condition 6 for a cut, let $\sigma_s(e_{\lambda_1})$ be an A-leader. Then $\sigma_s(q) = \sigma_v(q)$ by the definition of recursive expansion, and $\sigma_s(\phi(q)) = \sigma_v(\phi(q)) = d'$ because ϕ is a containment mapping. Because d and d' belong to the same component as d in the substitution graph of $\mathcal{Q}$, we are done. ∎

In [NaSa87], Naughton and Sagiv give the following four conditions, any one of which is individually sufficient for a linear sirup $\mathcal{Q}$ with an unbounded head chain to be unbounded:

A For no subset V of $\mathcal{D}_n$ does $\bigcup_{v \in V} s(v) = V$.

B $\mathcal{Q}$ contains no repeated EDB predicates.

C No persistent or stem variable appears in an EDB predicate of $\mathcal{Q}$.

D For no argument position s and two EDB subgoals e_i and e_j of $\mathcal{Q}$ does e_j have a persistent or stem variable at position s, while e_i has a variable belonging to some nondistinguished A at position s.

We claim that each of these four conditions individually implies that $\mathcal{Q}$ has no cut; the proofs in each case are not difficult:

A If for no subset V of $\mathcal{D}_n$ does $\bigcup_{v \in V} s(v) = V$, then the rule $\mathcal{Q}$ has no cycles in its substitution graph, and hence has no stem and no persistent variables. Since the cut condition requires at least one such variable, $\mathcal{Q}$ has no cut.

B Strictly speaking, we can ignore rules with no repeated EDB predicates because it is implicit in our viewpoint of linear sirups that they involve only one nonrecursive predicate name. However, it is useful to see how the Lemma 4.4 construction, even when presented with a Datalog rule with unique EDB predicate names, ensures that the corresponding linear sirup $\mathcal{Q} = (h, \mathcal{E})$ can have no cut. First, note that if $\phi(\mathcal{Q}_i(t_1, \lambda_1)) = \mathcal{Q}_i(t_2, \lambda_2))$ is part of a containment mapping ϕ for such a rule, then $\lambda_1 = \lambda_2$ is forced. If we suppose then that such a linear sirup $\mathcal{Q}$ had a (d, A)-cut e_j of some other subgoal e_i, then we see immediately by condition 4 for a cut that $e_i = e_j$. Since now it is impossible for part 3 for the cut condition to be fulfilled, we have a contradiction.

C If no persistent or stem variable appears in an EDB predicate of $\mathcal{Q}$, then it is impossible for condition 2 for a cut to be fulfilled.

D This final condition is the one which most closely approximates our cut condition. In fact, one sees that the stated condition is a slight weakening of axioms 2, 3, and 4 of the cut condition.

It should be observed that the conditions **A, B, C, D** are not incommensurate: in fact condition **A** implies condition **C**, while condition **C** in turn implies condition **D**, as has been observed by Naughton and Sagiv. The classes described by conditions **B** and **C** are incommensurate.

On the other hand, there are unbounded rules with unbounded head chains and no cut which satisfy none of the [NaSa87] conditions.

Example 4.18 The rule

$$1A3 \quad A22 \quad A32 \quad 332 \quad 322,$$

obtained by appending the distinguished variable 2 to each EDB subgoal of Naughton's rule, has an unbounded head chain, satisfies none of the four conditions **A-D** above, but has no cut. By Theorem 4.15, the rule is unbounded. ∎

4.5 Cut completion

We may arrive at a stronger polynomial-time testable version of the cut condition at the expense of a greater time complexity in testing for it as follows.

Again, let Q be a linear sirup of arity n with k EDB subgoals, and suppose that Q has an unbounded head chain. Let e_i and e_j be two EDB subgoals of Q, and suppose that e_j is a (d, A)-cut of e_i. We say that e_j is a *complete* (d, A)-cut of e_i if for every EDB subgoal e_k of Q that contains a variable belonging to A, there is a second EDB subgoal $e_{k'}$ (to be called a (d, A)-*cut completion* of e_k) that satisfies the following two properties:

1. If $\sigma_s(e_k)$ belongs to A and has maximal height amongst the variables of e_k that belong to A, then $\sigma_s(e_{k'})$ belongs to the same component as d in the substitution graph of Q.

2. If $\sigma_s(e_k)$ belongs to A and does not have maximal height amongst the variables of e_k that belong to A, then $\sigma_s(e_{k'})$ belongs to some nondistinguished variable C.

Theorem 4.19 *If $\mathcal{Q}$ is uniformly bounded and has an unbounded head chain, then $\mathcal{Q}$ has a complete cut.*

Proof By Theorem 4.15, $\mathcal{Q}$ has a cut. Suppose then that the EDB subgoal e_{λ_2} is a (d, A)-cut of some other EDB subgoal e_{λ_1} of $\mathcal{Q}$. It will be our assumption that the e_{λ_2}-e_{λ_1} cut arises exactly in the way described in our proof to Theorem 4.15—that is, we shall assume that there are recursive expansions $\mathcal{Q}_i$ and $\mathcal{Q}_j$ of $\mathcal{Q}$ with $i < j$ and a containment mapping $\phi : \mathcal{Q}_i \rightarrow \mathcal{Q}_j$ such that $q = \mathcal{Q}_i(t_1, \lambda_1)$ is the leftmost EDB subgoal on the head chain of $\mathcal{Q}_i$ that contains some nondistinguished variable A^l at an argument position s that maps to a persistent distinguished variable d', and where $\phi(q) = \mathcal{Q}_j(t_2, \lambda_2)$ for some t_2. Also, again as in Theorem 4.15, we may additionally assume that $i \geq 3n$, that $j \geq 4nk(i-1)$, and finally that $\phi(\mathcal{Q}_i(i-\alpha, \lambda)) = \mathcal{Q}_j(j-\alpha, \lambda)$ for all λ when α is in the range $0 \leq \alpha \leq n$. It follows therefore that $t_1 \leq i - n$ here.

In the proof of Theorem 4.15, we used ϕ, q, A^l, and d' to deduce that e_{λ_2} must be a (d, A)-cut of e_{λ_1} for some variable d in the same component as d' of the substitution graph of $\mathcal{Q}$; now, it will be our concern to show how the cut completion axioms 1 and 2 can additionally be deduced for the present set-up.

Suppose that e_λ is an EDB subgoal of $\mathcal{Q}$ that contains at least one variable belonging to A, and suppose that s_0 is chosen so that $\sigma_{s_0}(e_\lambda)$ is a variable of maximal height amongst all variables of e_λ that belong to A. We must show that e_λ has cut completion $e_{\lambda'}$ in $\mathcal{Q}$. If the A-leaders of $\mathcal{Q}$ have height β, then we can write

$$\beta = \text{height}(\sigma_{s_0}(e_\lambda)) + \tau$$

for some τ in the range $0 \leq \tau \leq n$, so that

$$\sigma_{s_0}(\mathcal{Q}_i(t_1 - \tau, \lambda)) = A^l = \sigma_s(\mathcal{Q}_i(t_1, \lambda_1)) = \sigma_s(q)$$

by the definition of recursive expansion[1]. Now suppose that

$$\phi(\mathcal{Q}_i(t_1 - \tau, \lambda)) = \mathcal{Q}_j(t_2, \lambda').$$

We claim that $e_{\lambda'}$ is a (d, A) cut completion of e_λ in $\mathcal{Q}$.

[1]It may be asked how we know that $t_1 - \tau \geq 0$. The answer is that because q contains an A-leader at position s, we know that t_1 is as large as height(A-leader) + l, while τ is *smaller* than height(A-leader).

In proof, we turn first to the first cut completion axiom, which is easily verified: because

$$\sigma_{s_0}(\mathcal{Q}_i(t_1 - \tau, \lambda)) = A^l,$$

and we know that $\dot{\phi}(A^l) = d'$, which belongs to the same component as d in the substitution graph of $\mathcal{Q}$, we see that $e_{\lambda'}$ must also have a distinguished variable from the same component as d at position s_0, in order for d' to arise at position s_0 in $\mathcal{Q}_j(t_2, \lambda')$.

To verify the second cut completion axiom, let $\sigma_{s_1}(e_\lambda)$ be a variable that belongs to A, yet $\sigma_{s_1}(e_\lambda)$ does not have maximal height amongst all the variables of e_λ that belong to A. Then we can write

$$\text{height}(\sigma_{s_0}(e_\lambda)) = \tau_0 - \text{height}(\sigma_{s_1}(e_\lambda))$$

for some $0 < \tau_0 \leq n$. Now suppose that contrary to the second cut completion axiom, we find that $\sigma_{s_1}(e_{\lambda'})$ does not belong to a nondistinguished variable, but is on the contrary a persistent or stem variable. Because

$$\sigma_{s_0}(\mathcal{Q}_i(t_1 - \tau, \lambda)) = A^l = \sigma_{s_1}(\mathcal{Q}_i(t_1 - \tau - \tau_0, \lambda)),$$

we see that

$$\sigma_{s_1}(\mathcal{Q}_i(t_1 - \tau, \lambda)) = A^{l+\tau_0}$$

is a nondistinguished variable of $\mathcal{Q}_i$ that $\dot{\phi}$ maps to a persistent distinguished variable. But because the symbol $A^{l+\tau_0}$ also occurs in $\mathcal{Q}_i(t_1 + \tau_0, \lambda)$, to the left of q in the query tableau representation of $\mathcal{Q}$, we have a contradiction, by our choice of q.

∎

In the first line of the figure below, we rewrite Naughton's rule yet again. Underneath each EDB subgoal e_k that contains a variable belonging to A, we have written a cut completion $e_{k'}$ corresponding to the $(3, A)$-cut of e_1 by e_2. Therefore Naughton's rule has a complete cut, in accordance with Theorem 4.19.

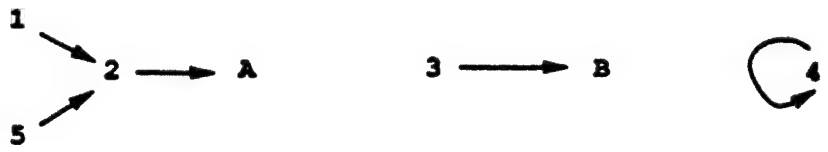
1A3	A2	A3	33	32
	A3	33		33

We conclude this chapter with the study of a more complex example.

Example 4.20 Consider the linear sirup $\mathcal{Q}$

2AB42 2A2 1A3 411 4A3 445 444

of arity $n = 5$ with $k = 6$ EDB subgoals. The substitution graph of Q is



and one can easily check by hand (or verify using Naughton's algorithm) that Q has an unbounded head chain. The variable 4 is the only persistent variable, while the variables 1, 2, 5, and A belong to A, and 3 and B belong to B. The variables 1 and 5 are both A-leaders, and each has height two; 3 is the only B-leader, and it has height one. There are no stem variables.

Now, we look to see what cuts we can find in Q. First, we note that $e_1 = 2A2$ can be cut by no other EDB subgoal, because it contains no leader. Next, we turn to $e_2 = 1A3$, which contains the two leaders $\sigma_1(e_2) = 1$, and $\sigma_3(e_2) = 3$. Thus, it is possible that there may be either a $(4, A)$-cut of e_2 by some other EDB subgoal, or alternatively a $(4, B)$-cut. We consider the latter possibility first. First, if e_k is a $(4, B)$-cut of $1A3$, then by Axiom 3 for a cut, above, we have $\sigma_3(e_k) = 4$. The only EDB predicate with a 4 in position 3 is $e_6 = 444$, but unfortunately, e_6 also has a 4 in position 2, in which position e_2 has the nonpersistent, nonleader variable A. Thus Axiom 5 for a cut would be violated, so we conclude there is no $(4, B)$-cut of e_2.

Next, we return to the possibility that there is a $(4, A)$-cut of e_2 in Q. Here, we meet with initial success: the subgoal $e_4 = 4A3$ is a $(4, A)$-cut of e_2. Also, we find that the subgoals e_3 through e_5 have valid completions:

$$\begin{array}{ccccccc} 2AB42 & 2A2 & 1A3 & 411 & 4A3 & 445 & 444 \\ & & 4A3 & 444 & 445 & 444 & \end{array}$$

but then we find ourselves in a quandary with e_1. A $(4, A)$ cut completion e_k of e_1 must have 4 in positions 1 and 3, and a nonpersistent variable in position 2. Because there is no such EDB subgoal, we find that the $(4, A)$-cut of e_2 by e_4 is not complete.

So far, we have ruled out the existence of complete cuts for e_1 and for e_2. Are there complete cuts for any of the remaining four EDB subgoals e_3, e_4, e_5, or e_6? We can exclude e_6 immediately, because it contains no leader. The subgoal e_3, however, contains two occurrences of the A-leader variable 1. We may hope therefore for a $(4, A)$-cut, but we

already know that e_1 has no $(4, A)$-cut completion, so there is no point in continuing. Turning next to e_4, we find that it contains the B-leader 3 at position 3. Is there a $(4, B)$-cut of e_4 in $\mathcal{Q}$? The answer is no—the argument may be carried out just as we did when we considered a $(4, B)$-cut for e_2, above. Finally, what about e_5? The variable 5 at position 3 in e_5 is an A-leader, and $e_6 = 444$ is a $(4, A)$-cut of e_5, but again, because such a cut has no completion, we find our path is blocked.

In short, $\mathcal{Q}$ contains no complete cut whatsoever. Our final conclusion (via Theorem 4.19): $\mathcal{Q}$ is unbounded. ∎

Chapter 5

Rule factorization

In this chapter, we shall examine the problem of rule factorization. Just as in our previous chapters, we shall be restricting our attention to finite nonempty rule sets S whose members are linear recursive Datalog rules over a single recursive predicate p.

5.1 Factorization and the equation $r = r_1 r_2$

We recall some definitions. We define linear recursive Datalog rules r_1 and r_2 to be *equivalent* (and write $r_1 \simeq r_2$) if they simultaneously stand in the relationships $r_1 \succeq r_2$ and $r_2 \succeq r_1$. Two rules r_1 and r_2 are said to be *isomorphic* (and we write $r_1 = r_2$) if r_1 and r_2 are identical up to a renaming of the their nondistinguished variables [Naug86b]. If r is the rule obtained by syntactically expanding the rule r_1 by the rule r_2, then r and $r_1 r_2$ are isomorphic and we shall say that r *admits the factorization* $r = r_1 r_2$.

In its most general form, the *rule factorization problem* can be simply stated: "Given a rule r, how can we find r_1 and r_2 so that $r = r_1 r_2$?" Because most of our theorem statements from Chapter 3 assumed such factorizations to be given, the rule factorization problem is basic to the applicability of our techniques.

Intuitively, we may think of rule factorization as "the inverse of recursive expansion," but there are some important technical points to be considered if we are to think of factorization in these terms. Most importantly, note that the factorization equation $r = r_1 r_2$ is to be interpreted in $\dot{\mathcal{J}}(S)$, where $S = \{r_1, r_2\}$, and *not* in the rule expansion semigroup $\mathcal{J}(S)$ itself. Because the rules r and $r_1 r_2$ in a factorization $r = r_1 r_2$ are identical up to the renaming of nondistinguished variables, it follows easily that $r \simeq r_1 r_2$, but the converse is not true in

general. The difficulty arises because *equivalence does not imply isomorphism*. Studying the factorization equation $r = r_1 r_2$ is therefore different from studying the algebraic relationship $r \simeq r_1 r_2$. The distinction arises because general rules may involve redundant EDB subgoals.

Example 5.1 The three rules

$$\begin{aligned} r &: \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(AB)} \ \& \ \mathtt{e(X_2X_1)} \ \& \ \mathtt{e(X_1X_2)} \ \& \ \mathtt{e(X_2X_1)} \\ r_1 &: \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_2X_1)} \ \& \ \mathtt{e(X_2X_1)} \\ r_2 &: \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(AB)} \ \& \ \mathtt{e(X_2X_1)} \end{aligned}$$

stand in the relationship $r \simeq r_1 r_2$, but r is not isomorphic to $r_1 r_2$ because r contains a redundant occurrence of the EDB subgoal $\mathtt{e(X_2X_1)}$, while $r_1 r_2$ does not. Strictly speaking, the equivalence $r \simeq r_1 r_2$ is therefore not a factorization (although it is certainly an algebraic relationship, and we could make it into a factorization by deleting one of the redundant EDB subgoals from r). ∎

To summarize, we might say "factorization is defined relative to isomorphism, and not equivalence." There are both advantages and disadvantages with taking this viewpoint. On the positive side, factorization is perhaps more naturally thought of as the inverse of recursive expansion than it is in terms of the algebraic relationship $r \simeq r_1 r_2$, and we shall be able to find efficient factorization algorithms. On the other hand, an unfortunate consequence of our definition is that whether a nontrivial factorization exists for an element in a $\simeq$ equivalence class of $\hat{\mathcal{J}}(S)$ may depend on the particular rule expansion r chosen from that class.

Example 5.2 The three rules

$$\begin{aligned} r &: \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_2X_1)} \ \& \ \mathtt{e(AC)} \ \& \ \mathtt{e(AB)} \ \& \ \mathtt{e(X_2B)} \\ r_1 &: \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_2X_1)} \ \& \ \mathtt{e(X_2A)} \\ r_2 &: \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1X_2)} \ \& \ \mathtt{e(X_1B)} \ \& \ \mathtt{e(X_1C)} \end{aligned}$$

stand in the relationship $r \simeq r_1 r_2$, but r cannot be written isomorphically as a recursive expansion $r = r'r''$ of two rules r' and r'' with at least one EDB subgoal apiece. (See the following two sections for additional explanation and proof method). However, we can write $r \simeq r_0 = r_1 r_2$, where r_0 is the rule

$$r_0 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_2X_1)} \ \& \ \mathtt{e(X_2B)} \ \& \ \mathtt{e(X_2C)} \ \& \ \mathtt{e(X_2A)}.$$

∎

Another point we must consider before we attempt to develop a factorization theory is that a factorization may be "degenerate" in the sense that one of the two factors r_1 or r_2 may act as an identity element:

Example 5.3 The simple transitive closure rule

$$r : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \mathtt{e(AX_2)}$$

admits the factorization $r = r_1r_2$, where the two rules r_1 and r_2 are defined as follows:

$$r_1 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \mathtt{e(AX_2)}$$
$$r_2 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1X_2)}$$

Here, r_2 is essentially "the identity," while r_1 is just r itself. A good factorization theory should identify such factorizations as trivial in some sense. ∎

To sidestep these two difficulties, we shall take the definition that if r is a rule with $k \geq 2$ EDB subgoals and r_1 and r_2 each have at least one nonrecursive subgoal apiece, then we shall call a factorization $r = r_1r_2$ *nontrivial*; otherwise, the factorization is said to be *trivial*. The definition not only ensures that degenerate factorizations involving rules with no EDB predicates will be identified, but also that every rule, even one with redundant EDB predicates, will have a factorization into *irreducible* rules, as described in the next section.

5.2 Irreducible rules

When a rule r admits no nontrivial factorization $r = r_1r_2$, we call it *irreducible*.

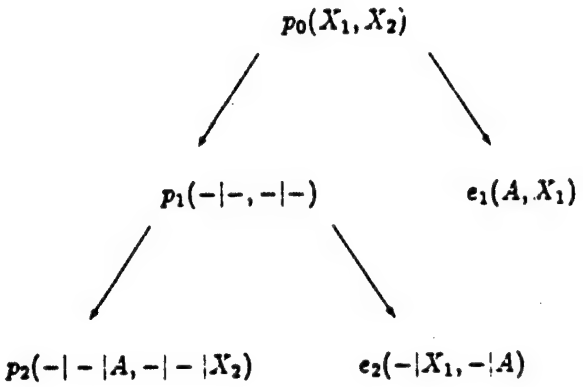
Example 5.4 The rule

$$r : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(AX_2)} \ \& \ \mathtt{e(AX_1)} \ \& \ \mathtt{e(X_1A)}$$

is irreducible. In proof, suppose to the contrary that there is a nontrivial factorization $r = r_1r_2$. Then r_1 and r_2 each have the form

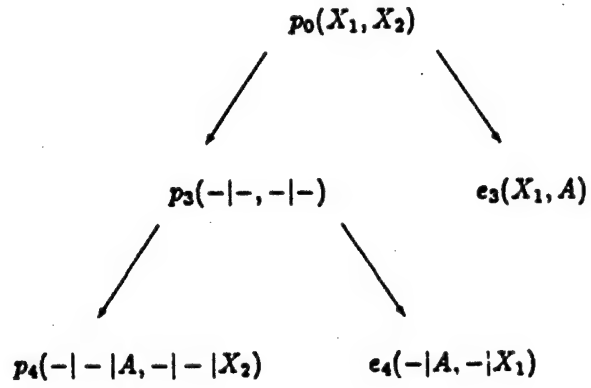
$$\mathtt{p(X_1X_2)} \ :- \ \mathtt{p(--)} \ \& \ \mathtt{e(--)},$$

and we must decide how to "fill in the blanks" so that r and r_1r_2 are isomorphic. There are two possible expansion tree forms for such a factorization. The first possibility is that we have an expansion tree of the form



where the $\mathbf{e(AX_1)}$ of r occurs in r_1, and $\mathbf{e(X_1A)}$ arises upon expansion of r_1 by r_2. (We have written the expansion tree with unsimplified and as yet partially indeterminate substitution chains, and we have put subscripts on the various occurrences of the predicate names e and p in the tree only in order to distinguish between them more easily).

The other possibility is that we have an expansion tree of the form



where the roles of $e(\mathtt{A}\tilde{\mathtt{x}}_1)$ and $e(\mathtt{X}_1\mathtt{A})$ are reversed.

To see why neither expansion tree form above can correspond to an actual recursive expansion of rules, we focus on the subgoal substitutions of r_1. Inspecting either p_2 and e_2 from the first expansion tree, or alternatively p_4 and e_4 from the second, we see in each case that r_1 must have three subgoal substitutions $\langle -|A, -|X_2, -|X_1\rangle_{r_1}$. Because the three blanks to be filled in in the last expression must be occupied by three *distinct* distinguished variables, we have a contradiction—the arity of p is only two, so r_1 has just *two* distinguished variables. Therefore r is irreducible.

∎

We can generalize the notion of rule factorization in the natural way to consider factorizations into more than two rules. By a simple inductive argument we can show:

Theorem 5.5 *Every linear Datalog rule*

$$r : \mathtt{p} \ :- \ \mathtt{p} \ \& \ e_1 \ \& \ \cdots \ \& \ e_k$$

is either irreducible or is expressible as a factorization $r = r_1 r_2 \cdots r_s$ of irreducible rules r_i for some $s \geq 2$.

Proof We use induction on k, the number of EDB subgoals that occur in r. The theorem is clearly true if $k = 1$. Assume it is true for every integer $k < N$. Then if r is not irreducible we may write $r = r'r''$ where r' and r'' each have at least one, and fewer than k, EDB subgoals apiece. Hence the inductive hypothesis applies, and both r' and r'' are either irreducible or are themselves expressible as a product of irreducibles. Therefore r too is a product of irreducibles, as required. ∎

We pause again to look at another example.

Example 5.6 The "same generation query" [RSUV89]

$$r : \mathtt{p}(\mathtt{X}_1\mathtt{X}_2) \ :- \ \mathtt{p}(\mathtt{AB}) \ \& \ e(\mathtt{X}_1\mathtt{A}) \ \& \ e(\mathtt{X}_2\mathtt{B})$$

can be factored into irreducibles as $r = r_1 r_2$, where

$$r_1 : \mathtt{p}(\mathtt{X}_1\mathtt{X}_2) \ :- \ \mathtt{p}(\mathtt{X}_1\mathtt{A}) \ \& \ e(\mathtt{X}_2\mathtt{A})$$

and

$$r_2 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(AX_2)} \ \& \ \mathtt{e(X_1A)}$$

are commuting rules that we have met before. In fact, r_1 is precisely the rule d from Example 3.4, while r_2 appears in Examples 2.8 and 3.5. ∎

It is our belief that most "natural" linear recursions will factor in simple ways; of course, there is no ultimately decisive way that we can defend our claim. But the last example query is perhaps one point in our favor. Although the same generation query is often put forward as a "nontrivial" recursion (in particular because it is "2-sided" in the sense of Naughton), we find that it factors naturally as the product of two 1-sided recursions that themselves play roles in the generation of classes of recursions to which our semigroup techniques have been seen to apply. Thus it is perhaps not unreasonable to expect that more complex rule sets will also typically involve recursions that are built from simpler rules by recursive expansion.

It should be said, however, that the particular $r = r_1 r_2$ factorization of the same generation rule in Example 5.6 does not have immediate implications for its efficient evaluation. In particular, the commutative decomposition does not improve upon, or indeed say anything directly about the "counting" algorithm [Ban86a] or other algorithms for same generation rule evaluation. However, two points can be made. First, the factorization $r = r_1 r_2$ does reveal some information about the structure of the same generation rule that can illuminate the process of writing similar logical rules (see Section 5.5). Second, every factorization $r = r_1 r_2$ has the potential to reveal redundancy in settings where r appears in conjunction with other rules (which themselves, we hope, may also admit factorizations into parts involving r_1 and r_2 so that our methods in Chapter 3 will apply).

Next, we shall take up the factorization problem from a constructive point of view.

5.3 Bounded arity factorization

Example 5.4 contains the germ of an idea for an algorithm that factors a *bounded arity* linear recursive rule in time polynomial in the size of the rule.

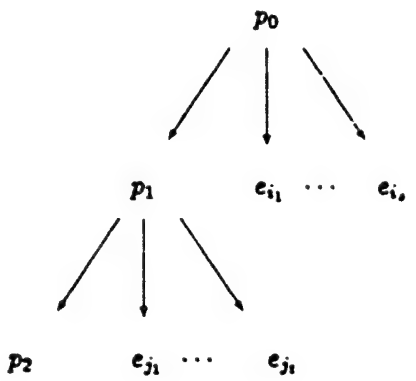
Theorem 5.7 *Suppose r is a linear recursive Datalog rule whose recursive arity n does not exceed a fixed positive constant K. Then r can be either recognized as irreducible or*

constructively factored as a nontrivial product $r = r_1 r_2$ in time $O(N^{2(K-1)})$, where N is the size of the rule in standard Datalog notation.

Proof We shall assume that the rule r takes the form

$$r : \mathtt{p} \ :- \ \mathtt{p} \ \& \ \mathtt{e}_1 \ \& \ \cdots \ \& \ \mathtt{e}_k$$

with recursive arity $n \leq K$ and k EDB subgoals. We may view the problem of factoring r as one of building an abstract expansion tree $r_1 r_2$ of the form



where $s, t \geq 1$, $s + t = k$, and r and $r_1 r_2$ are identical up to renaming of their nondistinguished variables and the rearrangement of their subgoals. Again, we have put subscripts on the three occurrences of p in the expansion tree only in order to distinguish between them more easily.

Our method will take the following form: first, we shall present a nondeterministic algorithm for the factorization problem, and then we shall use the bounded arity assumption to show how the algorithm may be made to be deterministic and still run in polynomial time.

Suppose for the moment that we are given a guessed *trial correspondence mapping* ψ between the subgoals of r and the subgoals of the expansion tree $r_1 r_2$. We shall require the mapping ψ to be (in part) a bijection between the k EDB subgoals $\{e_1, e_2, \ldots, e_k\}$ of r and the k EDB subgoals $e_{i_1}, \ldots, e_{i_s}, e_{j_1}, \ldots, e_{j_t}$ of the expansion tree $r_1 r_2$ such that e_i and $\psi(e_i)$ are identical up to a fixed renaming of nondistinguished variables that is independent

of the choice of i. We must require the recursive subgoal correspondence $\psi(p) = p_2$ between r and r_1r_2 to respect the fixed nondistinguished variable renaming as well. Intuitively, the mapping ψ simply specifies for each e_i whether its syntactic counterpart in r_1r_2 is to be found at the first (corresponding to r_1) level of the tree, or alternatively the second (corresponding to recursive expansion of r_1 by r_2).

Having guessed the correspondence ψ, we wish to know: "can the partial expansion tree corresponding to ψ be completed to conform to an actual expansion of two rules r_1r_2?" We saw in Example 5.4 how a simple *necessary* condition for the completion of the tree may be given by counting subgoal substitutions of r_1. More precisely, for every distinct distinguished variable X_a that occurs either in p_2 or in some $e_{j\beta}$, we saw that r_1 must have a subgoal substitution of the form $\langle -|X_a\rangle_{r_1}$. We begin therefore by counting the number c' of distinguished variables that occur either in p_2 or in some $e_{j\beta}$. Next, if A is a nondistinguished variable that occurs either in p_2 or in some $e_{j\beta}$, and A *also* occurs in some $e_{i\gamma}$, then we know that p_1 must contain an occurrence of A, and that r_1 has a subgoal substitution of the form $\langle -|A\rangle_{r_1}$. Let the number of such nondistinguished variables A be c''. The condition from Example 5.4 was then exactly this: the sum $c = c' - c''$ can be no greater than the recursive arity n of the rule r, for otherwise the expansion tree cannot exist.

In fact, we claim that the stated condition is also *sufficient* for the completion of a trial correspondence as well. In other words, we claim: a trial correspondence ψ will correspond to an actual recursive expansion r_1r_2 if and only if we have $c' + c'' \leq n$.

The claim is easily proved by direct construction of the rules r_1 and r_2. Suppose that the $c = c' + c''$ variables accounted for in the two counts are c' distinguished variables $X_{a_1}, \ldots, X_{a_{c'}}$ and c'' nondistinguished variables $A_1, \ldots, A_{c''}$. We shall call the $c' - c''$ variables

$$\{X_{a_1}, \ldots, X_{a_{c'}}, A_1, \ldots, A_{c''}\}$$

the *charged variables* of the trial correspondence ψ.

We now turn to the construction of r_1. Because $c' + c'' \leq n$, we may construct the first $c' + c''$ positions in the recursive subgoal of r_1 by placing all the charged variables $X_{a_1}, \ldots, X_{a_{c'}}, A_1, \ldots, A_{c''}$ in the first $c' + c''$ variable positions. Any remaining variable positions of the recursive subgoal of r_1 are filled in identically by putting the variable X_i at every position i, where $c' + c'' < i \leq n$. The nonrecursive subgoals of r_1 are made to match their trial correspondence counterparts in r exactly (recall that no subgoal substitutions are

applied to these subgoals when r_1 is expanded by r_2). So far, we have forced the expansion tree to agree with r at its top level.

Now, what about r_2? We turn first to the construction of its nonrecursive subgoals. At positions where charged variables are to occur in $r_1 r_2$, we must be certain that the appropriate substitution $\langle X_i | X_{a_i} \rangle_{r_1}$ or $\langle X_i | A_i \rangle_{r_1}$ takes place; thus the correct X_i must be placed at every such position in the EDB subgoals of r_2. Any remaining variable positions in the nonrecursive subgoals of r_2 are necessarily occupied by nondistinguished variables that appear in no $e_{i,\gamma}$. If W is such a variable, we can replace all occurrences of W in p_2 and the nonrecursive subgoals of r_2 by the symbol B' for some implicit substitution $\langle B | B' \rangle_{r_1}$ of r_1 and *new* nondistinguished variable symbol B. (A nondistinguished variable renaming $W \rightarrow B'$ has taken place here). If the symbol W (now renamed B') occurs somewhere in p_2, say at position l, then $\langle X_l | B \rangle_{r_2}$ becomes a substitution of r_2, and the unsimplified substitution chain at position l in p_2 has the form $\langle X_l | B | B' \rangle_{r_1 r_2}$. The remaining substitutions of r_2 are determined by the substitutions of r_1 in a similar manner—if the partial substitution chain at position l in p_2 is $\langle - | X_{t_1} | X_{t_2} \rangle_{r_1 r_2}$, then $\langle X_l | X_{t_1} \rangle_{r_2}$ becomes a substitution of r_2. There can be no possible conflict between such definitions because each r_2 subgoal substitution is applied only once in the expansion tree (in the appropriate position of p_2.)

We have proved our claim, above, but the accomplishment is a modest one. All we know is that if someone suggests to us how to split the EDB subgoals of r between r_1 and r_2 by giving us a trial correspondence ψ, then we can determine whether the suggestion corresponds to an actual recursive expansion, and we can construct a corresponding expansion tree, if one exists. But because there are essentially on the order of 2^k possible trial correspondences to be considered, we do not yet have an efficient algorithm for the bounded arity factorization problem.

We shall overcome the difficulty by guessing not the trial correspondence itself, but instead merely the *identities of its charged variables.*

Recall that the *linking graph* G of r is an undirected graph on the subgoal vertices $p, e_1, \ldots, e_k$, with an edge between two subgoals if they share a nondistinguished variable. Here, we shall introduce a *colored linking multigraph* Γ of r whose edges are again determined by shared nondistinguished variables, but where these edges are also *colored* according to the particular nondistinguished variable shared. Thus between two vertices of the multigraph we may have several edges, one of each of several colors, each color corresponding to some

applied to these subgoals when r_1 is expanded by r_2). So far, we have forced the expansion tree to agree with r at its top level.

Now, what about r_2? We turn first to the construction of its nonrecursive subgoals. At positions where charged variables are to occur in $r_1 r_2$, we must be certain that the appropriate substitution $\langle X_i, X_{a_i} \rangle_{r_1}$ or $\langle X_i, A_i \rangle_{r_1}$ takes place; thus the correct X_i must be placed at every such position in the EDB subgoals of r_2. Any remaining variable positions in the nonrecursive subgoals of r_2 are necessarily occupied by nondistinguished variables that appear in no $e_{i,j}$. If W is such a variable, we can replace all occurrences of W in p_2 and the nonrecursive subgoals of r_2 by the symbol B' for some implicit substitution $\langle B, B' \rangle_{r_1}$ of r_1 and *new* nondistinguished variable symbol B. (A nondistinguished variable renaming $W \rightarrow B'$ has taken place here). If the symbol W (now renamed B') occurs somewhere in p_2, say at position l, then $\langle X_l, B \rangle_{r_2}$ becomes a substitution of r_2, and the unsimplified substitution chain at position l in p_2 has the form $\langle X_l, B, B' \rangle_{r_1 r_2}$. The remaining substitutions of r_2 are determined by the substitutions of r_1 in a similar manner—if the partial substitution chain at position l in p_2 is $\langle -, X_{t_1}, X_{t_2} \rangle_{r_1 r_2}$, then $\langle X_l, X_{t_1} \rangle_{r_2}$ becomes a substitution of r_2. There can be no possible conflict between such definitions because each r_2 subgoal substitution is applied only once in the expansion tree (in the appropriate position of p_2.)

We have proved our claim, above, but the accomplishment is a modest one. All we know is that if someone suggests to us how to split the EDB subgoals of r between r_1 and r_2 by giving us a trial correspondence ψ, then we can determine whether the suggestion corresponds to an actual recursive expansion, and we can construct a corresponding expansion tree, if one exists. But because there are essentially on the order of 2^k possible trial correspondences to be considered, we do not yet have an efficient algorithm for the bounded arity factorization problem.

We shall overcome the difficulty by guessing not the trial correspondence itself, but instead merely the ***identities of its charged variables.***

Recall that the *linking graph* G of r is an undirected graph on the subgoal vertices $p, e_1, \ldots, e_k$, with an edge between two subgoals if they share a nondistinguished variable. Here, we shall introduce a *colored linking multigraph* Γ of r whose edges are again determined by shared nondistinguished variables, but where these edges are also *colored* according to the particular nondistinguished variable shared. Thus between two vertices of the multigraph we may have several edges, one of each of several colors, each color corresponding to some

different nondistinguished variables the two subgoals share.

Let $C = \{X_{a_1}, \ldots, X_{a_{c'}}, A_1, \ldots, A_{c''}\}$ be a guessed set of charged variables corresponding to some trial correspondence between the EDB subgoals of r and those of $r_1 r_2$. We know that it can be assumed that $c' - c'' \leq n$ here. To verify the guess, we must check that the EDB subgoals of r can be split so as to yield exactly the charged variable set C. (Such a split is in fact exactly a trial correspondence τ). We may apply the following five step *Γ-multigraph algorithm*.

1. Check whether all distinguished variables of p are in C. If not, answer "no: guess invalid," and exit. Otherwise:
2. Delete all edges colored $A_1, A_2, \ldots, A_{c''}$ from Γ.
3. Call the resulting graph Γ'.
4. Compute the connected components $\Gamma'_1, \Gamma'_2, \ldots \Gamma'_\ell$ of $\Gamma' \setminus p$.
5. If $\delta > 1$ and $\Gamma' \setminus p$ has a component containing no uncharged distinguished variables, then answer "yes: verified guess;" otherwise, answer "no: guess invalid."

To see that our verification algorithm is correct, we need to show that it answers "yes" if and only if the EDB subgoals of r can be split so as to yield exactly the charged variable set C.

Suppose first that the EDB subgoals of r can be split into two nonempty sets, one of "top level" EDB subgoals $\{e_{i_1}, \ldots, e_{i_s}\}$, and one of "second level" EDB subgoals $\{e_{j_1}, \ldots, e_{j_t}\}$, with $s, t \geq 1$, $s + t = k$, and charged variable set C. Then by the definition of C and Γ', the set $\{e_{j_1}, \ldots, e_{j_t}\}$ of second level subgoals contains no uncharged distinguished variables, and there can be no multigraph edge of Γ' between an e_{j_β} and an e_{i_α}. Similarly, p can neither contain an uncharged distinguished variable nor share an edge with an e_{i_α}, and so p must belong to Γ' also. Therefore p and $\{e_{j_1}, \ldots, e_{j_t}\}$ together induce a union of connect components in Γ', these vertices do not include all of Γ' because there is at least one vertex e_{i_1} detached from these vertices, and the component containing e_{j_1} can contain no uncharged distinguished variables. Therefore our verification algorithm will answer "yes," as required.

Conversely, suppose that the algorithm answers "yes" and we need to verify that there is a split of the EDB subgoals of r with charged variable set C. Here, it will suffice to identify the set $\{e_{j_1}, \ldots, e_{j_t}\}$ with a particular connected component Γ'_1 of $\Gamma' \setminus p$. The

assumption that $\Gamma' \setminus p$ has at least $\delta \geq 2$ components ensures that there is at least one remaining EDB vertex in the components $\Gamma'_2, \Gamma'_3, \ldots, \Gamma'_\delta$. Therefore the subgoals of any component that contains no uncharged distinguished variables can be identified with the the set $\{e_{j_1}, \ldots, e_{j_t}\}$. However, the charged variable set C' corresponding to this trial correspondence is only necessarily *contained* in C and need not be C itself. Thus strictly speaking we have not verified our guess precisely, but because $|C'| \leq |C| \leq n$, the charged variable counting condition is satisfied for C'. So r is still factorizable, and we can answer "yes" for this guess and still have a correct algorithm.

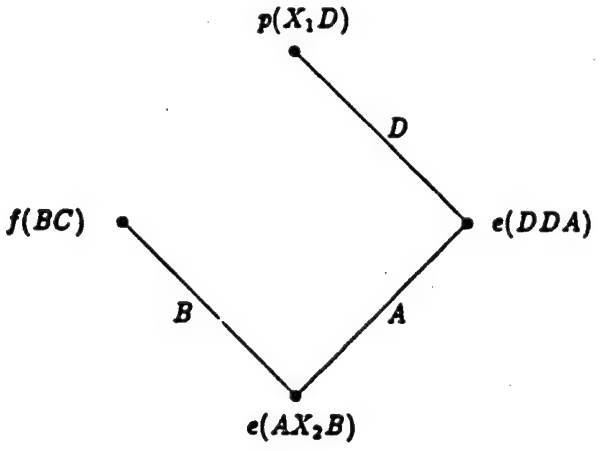
Because there are no more than $\binom{N}{K}^2$ charged variable set selections to be considered and the verification algorithm above can be made to run in time $O(N^2)$, our entire nondeterministic factorization algorithm made be made to be deterministic and run in time $O(N^{2(K+1)})$ by considering all possible charged variable sets in turn. ∎

There is no assertion in Theorem 5.7 that the two rules r_1 and r_2 are themselves irreducible. If a factorization into irreducible rules is desired, one can apply the theorem iteratively to obtain a factorization into irreducibles in time $O(N^{2(K+1)+1})$.

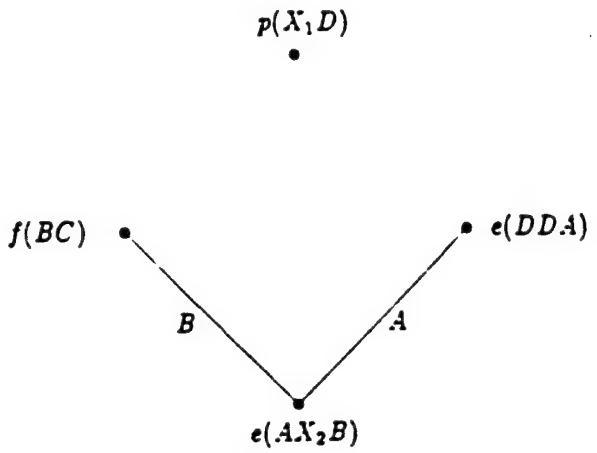
Example 5.8 For an example of how the Γ-multigraph algorithm works, consider the problem of factoring the rule

$$r : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1D)} \ \& \ \mathtt{e(DDA)} \ \& \ \mathtt{e(AX_2B)} \ \& \ \mathtt{f(BC)}.$$

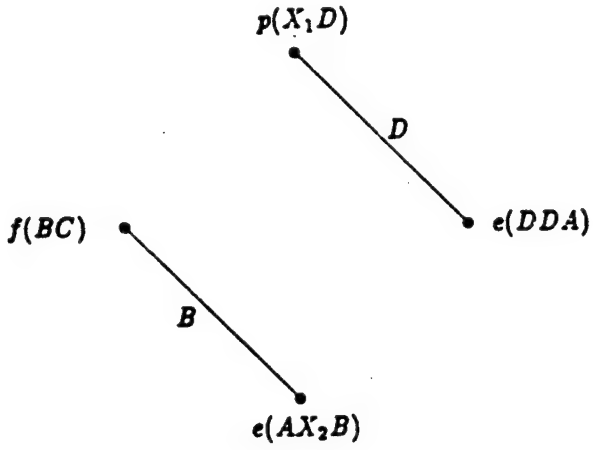
The colored linking multigraph of r is



If we make the (erroneous) guess of the charged variable set $C = \{X_1, D\}$, then the graph Γ' becomes



and $\Gamma' \setminus p$ is seen to have just $\delta = 1$ component. So the algorithm would answer "no: guess invalid" for this guess. However, r does have a factorization corresponding to the guessed charged variable set $\{X_1, A\}$. Here the graph Γ' becomes



we have $\delta = 2 > 1$ components in $\Gamma' \setminus p$, and the isolated vertex $e(DDA)$ in $\Gamma' \setminus p$ is a component containing no uncharged distinguished variables. Therefore the Γ-multigraph

algorithm would answer "yes: verified guess" for the guess $C = \{X_1, A\}$. Translating the guess into a factorization as suggested in the proof to Theorem 5.7, we obtain the rules

$$r_1 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \ \& \ \mathtt{e(AX_2B)} \ \& \ \mathtt{f(BC)}$$
$$r_2 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \mathtt{e(AAX_2)}$$

and one can check that r and $r_1 r_2$ are isomorphic. ∎

5.4 Unbounded arity factorization

If we do not know an explicit bound on the recursive arity of a linear rule r, then the Γ-multigraph factorization algorithm is not guaranteed to run in time polynomial in the size of r. The difficulty arises because the number of test (guessed) charged variable sets to be considered can no longer be guaranteed to be polynomially bounded in the input size.

Nevertheless, general (unbounded arity) linear recursive rules r may be either recognized as irreducible or constructively factored as a recursive expansion $r = r_1 r_2$ in polynomial time in the size of r by a more complex method, which we shall call the *flow graph algorithm*.

To describe the method we shall return to a rule notation similar to that used in Chapter 4, where we represented distinguished variables by positive integers and suppressed all predicate names. Because the factorizability of r is not influenced by the particular predicate names on subgoals but rather depends only on how variables are shared between subgoals, we can cast the irreducibility problem as an abstract decision problem in the following way.

RULE IRREDUCIBILITY

INSTANCE:

A collection $\mathcal{E}$ of $|\mathcal{E}| \geq 3$ finite subsets of $\mathcal{N} \cup \mathcal{A}$, where $\mathcal{N} = \{1, 2, 3, \ldots\}$ and $\mathcal{A} = \{A, B, C, \ldots\}$; also, a distinguished element $p \in \mathcal{E}$ and a *cost bound* c, which is an arbitrary positive integer.

QUESTION:

Can the collection $\mathcal{E} \setminus p$ be partitioned in 2 nonempty parts $\mathcal{U}$ and $\mathcal{D}$ with cost less than c? The cost of a $\mathcal{U}, \mathcal{D}$ partition is computed as the sum of all *variable charges*, a ollows:

(DISTINGUISHED VARIABLE CHARGES)

We charge 1 for every $n \in \mathcal{N}$ that occurs either in p or in some element of $\mathcal{D}$.

(NONDISTINGUISHED VARIABLE CHARGES)

We charge 1 for every letter $\alpha \in \mathcal{A}$ that occurs both in some element of $\mathcal{U}$, and in either p or in some element of $\mathcal{D}$.

We shall solve the rule irreducibility problem by reducing it to a certain flow problem in graphs. The method is most easily envisioned as a two-step process. In the first step, we shall reduce the irreducibility question to the following problem in bipartite graphs.

BIPARTITE DISCONNECT SET

INSTANCE:

A bipartite graph $G = (V, E)$ with vertex set $V = V_1 \cup V_2$, and $V_1 \cap V_2$ is empty. All edges $e \in E$ have one endpoint in V_1 and one in V_2. Also, two distinguished nodes s and t in V_1, and a cost bound c, an arbitrary positive integer.

QUESTION:

Can s be disconnected from t in G by removing no more than c vertices from V_2 only?

We shall first show how an instance of rule irreducibility of size n can be polynomially reduced to the problem of solving $O(n^2)$ instances of bipartite disconnect set. Then we shall show how each such bipartite disconnect set problem can be solved in polynomial time by flow techniques. Our final algorithm will have complexity $O(n^7)$, where again, n is the size of the input rule r in standard Datalog notation.

Let $\mathcal{E}$ be an instance of the rule irreducibility problem with distinguished element $p \in \mathcal{E}$ and cost bound c. We seek a partition of $\mathcal{E} \setminus p$ into nonempty parts $\mathcal{U}$ and $\mathcal{D}$ that has cost at most c.

Suppose for the moment that we *guess* from the set $\mathcal{E} \setminus p$ the identities of one element $u \in \mathcal{U}$ and one element $d \in \mathcal{D}$. It is our claim that the problem of deciding whether these initial guesses may be filled out into a complete partition of $\mathcal{E} \setminus p$ meeting the given cost bound can be thought of as an instance of bipartite disconnect set.

Here are the details. We build an instance of bipartite disconnect set as follows. First, we create a node $v_1 \in V_1$ for every element $e \in \mathcal{E}$, and we create a node $v_2 \in V_2$ for every symbol $n \in \mathcal{N}$ or $\alpha \in \mathcal{A}$ that occurs in some element of $\mathcal{E}$. Next, edges are drawn between

vertices $v_1 \in V_1$ and $v_2 \in V_2$ if and only if the symbol v_2 occurs in the set v_1. So far, we have a bipartite graph G on the disjoint vertex sets $V_1 \bigcup V_2 = V$. Each edge has one endpoint in V_1, and one in V_2.

Next, two new vertices s and t are added to V_1. The vertex s is made to be adjacent to each vertex of V_2 that either represents a distinguished variable $n \in \mathcal{N}$, or represents a nondistinguished variable occurring in u. The vertex t, on the other hand, is made to be adjacent to each vertex of V_2 that either occurs in p or d.

Our claim is this: the given starter guesses $u \in \mathcal{U}$ and $d \in \mathcal{D}$ can be filled out into a complete partition of $\mathcal{E} \setminus p$ that meets the given cost bound c if and only if in the graph G, the vertices s and t can be disconnected by the removal of at most c vertices from V_2.

We shall prove our claim below, but first it is useful to look at another example.

Example 5.9 Consider the problem of deciding whether the rule r given by

$$r : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \mathtt{e(A)} \ \& \ \mathtt{f(X_2B)} \ \& \ \mathtt{f(CX_2)} \ \& \ \mathtt{g(X_2AB)}$$

is irreducible. We first convert the problem into the simplified notation of the rule irreducibility problem statement, above. Replacing the distinguished variables by integers and suppressing all predicate names in every subgoal of r, we obtain the collection of sets $\mathcal{E} = \{e_1, e_2, e_3, e_4, e_5\}$ given by

$$\begin{aligned} p = e_1 &= \{1, A\} \\ e_2 &= \{A\} \\ e_3 &= \{2, B\} \\ e_4 &= \{2, C\} \\ e_5 &= \{2, A, B\}. \end{aligned}$$

The rule r will be factorizable if and only if the set $\mathcal{E} \setminus p = \{e_2, e_3, e_4, e_5\}$ can be partitioned into two nonempty parts $\mathcal{U}$ and $\mathcal{D}$ with a cost at most $c = 2$, the recursive arity of the rule r. In this particular example, there is such a partition—we may take for example $\mathcal{U} = \{e_4\}$, and $\mathcal{D} = \{e_2, e_3, e_5\}$, with the cost charge $|\{1, 2\}| = 2$ meeting the desired bound. However, note that some care must taken in choosing the desired partition—for example, the partition $\mathcal{U}' = \{e_5\}$, and $\mathcal{D}' = \{e_2, e_3, e_4\}$ would fail to meet the desired cost bound, because it has cost charge $|\{1, 2, A, B\}| = 4 > 2$.

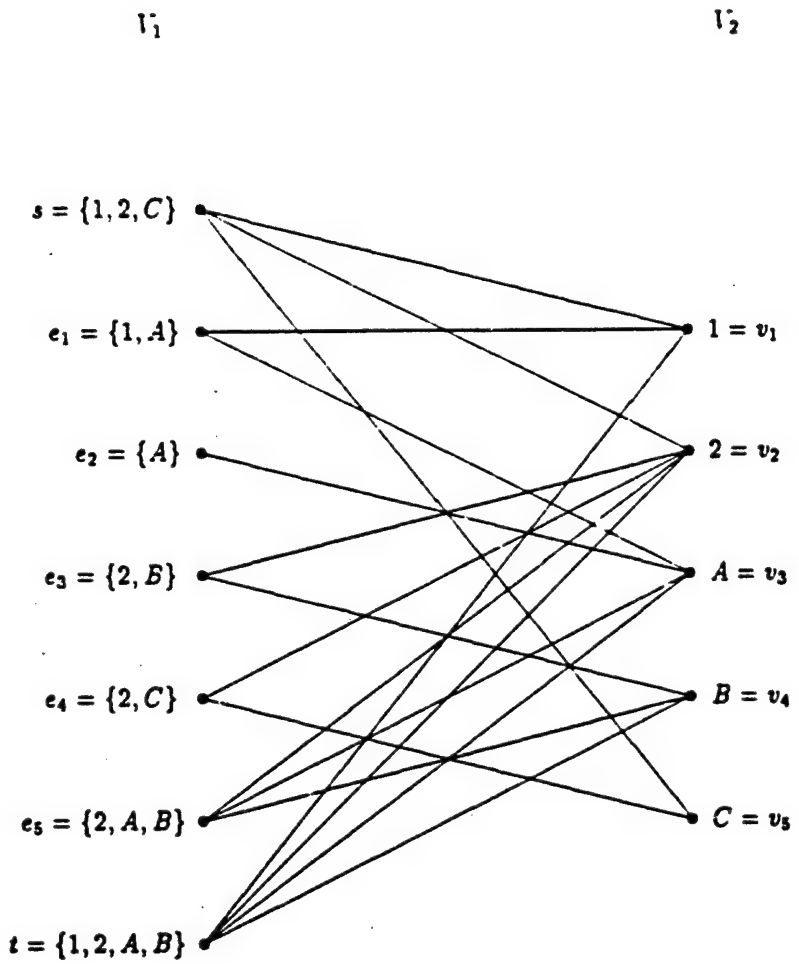


Figure 5.1: Bipartite disconnect set problem

Suppose we correctly guess two initial members of the former partition $u = e_4 \in \mathcal{U}$, and $d = e_5 \in \mathcal{D}$. It was our claim above that the question of whether this initial guess is extendable into a complete partition of $\mathcal{E} \setminus p$ that meets the desired cost bound can be thought of as an instance of the bipartite disconnect set problem. To illustrate the claim in the current example, in Figure 5.1 we have drawn the associated bipartite graph G, whose desired cost bound is also $c = 2$. Because the nodes s and t in V_1 can be disconnected by removing the two nodes 1 and 2 from V_2, the bipartite disconnect problem also has a "yes" answer. In fact, we can recover the partition $\mathcal{U}, \mathcal{D}$ by taking $\mathcal{D}$ to be all nodes of $\mathcal{E} \setminus p$ that are connected to t after the vertices 1 and 2 have been removed from G, while $\mathcal{U}$ is taken to be all the remaining nodes in $\mathcal{E} \setminus p$.

∎

Theorem 5.10 *Let $\mathcal{E} = \{p = e_1, e_2, \ldots, e_k\}$ be an instance of the irreducibility problem with cost bound c, and suppose that two distinct elements $u = e_i$ and $d = e_j$ are designated in $\mathcal{E} \setminus p = \{e_2, \ldots, e_k\}$. Let $G = G(\mathcal{E}) = (V_1 \cup V_2, E)$ be the corresponding constructed graph instance of the bipartite disconnect set problem.*

Then the following are equivalent:

(1). The vertices s and t can be disconnected in G by the removal of c vertices from V_2.

(2.) There is a partition of $\mathcal{E} \setminus p$ into two nonempty, disjoint parts $\mathcal{U}$ (containing u) and $\mathcal{D}$ (containing d) whose total variable charge is no more than c.

Proof Show that (1) ⇒ (2).

Suppose that the vertices s and t can be disconnected by the removal of the vertex set $C \subseteq V_2$ from G, and $|C| = c$. Let G' be the resulting subgraph of G. Then we shall construct a $\mathcal{U}, \mathcal{D}$ partition of $\mathcal{E} \setminus p$ meeting the desired conditions as follows. We let

$$\mathcal{D} = d \cup \{v \mid v \in \mathcal{E} \setminus p \text{ is connected to } t \text{ in } G'\},$$

and

$$\mathcal{U} = u \cup \{\text{Everything else in } \mathcal{E} \setminus p\}.$$

We must verify that the sets $\mathcal{U}$ and $\mathcal{D}$ are indeed a disjoint partition of $\mathcal{E} \setminus p$ whose total variable charge is at most c.

First, because $u \in \mathcal{U}$ and $d \in \mathcal{D}$, we see that $\mathcal{U}$ and $\mathcal{D}$ are nonempty, and by construction $\mathcal{U}$ and $\mathcal{D}$ together exhaust $\mathcal{E} \setminus p$. Next, we must show that $\mathcal{U}$ and $\mathcal{D}$ are disjoint. By construction of the graph G, if x is any node in G', then the following implications are valid.

1. Vertex x is reachable from d in $G' \Rightarrow x$ is reachable from t in G'.

2. Vertex x is reachable from u in $G' \Rightarrow x$ is reachable from s in G'.

Therefore $\mathcal{U} \cap \mathcal{D}$ must be empty, for if x is in $\mathcal{U} \cap \mathcal{D}$, then x is connected to t and to s, contradicting the fact that s and t are disconnected in G'.

To finish, we shall show that the charged variables of this $\mathcal{U}, \mathcal{D}$ partition are all necessarily elements of C. We shall consider charged distinguished, and then nondistinguished, variables in turn.

Let n be a distinguished variable charged to the $\mathcal{U}, \mathcal{D}$ partition. Then n either occurs in p or in some element of $\mathcal{D}$. If n is not a removed vertex, (i.e., an element of C), then n is a vertex of V_2 in G' that serves to connect s (which, recall, has all *all* distinguished variable adjacencies) either to t, or to a vertex connected to t. Thus we would conclude that s is connected to t in G', a contradiction.

Finally, suppose that A is a nondistinguished variable charged to the $\mathcal{U}, \mathcal{D}$ partition. Then A appears both in some $u' \in \mathcal{U}$, and also in some $d' \in \mathcal{D}$. Thus if the variable A is not one of the deleted vertices C, we conclude that $\mathcal{U}$ and $\mathcal{D}$ are connected, which is again a contradiction by the construction of $\mathcal{U}$ and $\mathcal{D}$.

Show that (2) $\Rightarrow$ (1).

Let $\mathcal{U}, \mathcal{D}$ be a partition of $\mathcal{E} \setminus p$ such that $u \in \mathcal{U}$ and $d \in \mathcal{D}$. Let the charged variable set corresponding to this partition be C. We claim that removing the vertices C from V_2 in G must disconnect s from t in the resulting graph G'.

In proof, suppose to the contrary that there is simple path from t to s in G'. Vertices of V_2 that appear along this path are necessarily uncharged variables. In fact, we claim slightly more is true: *Any vertex of V_2 on a path from t to s in G' must be an uncharged nondistinguished variable.* Why? First, it is clear that no vertex at distance one from t in G' can be a distinguished variable, because such a variable must appear either in p or in d, so therefore is charged and must have been removed from G'. If some vertex in V_2 at a distance greater than 1 from t along the path were a distinguished variable, then we could choose

the closest such variable, say $v_2 \in V_2$, at distance $\delta \geq 2$ from t. By assumption, no vertex of V_2 at distance $< \delta$ from t is a distinguished variable, and we know each is uncharged. Therefore each vertex $v_1 \in V_1$ before v_2 on the path from t to s must correspond either to p, or to an element of $\mathcal{D}$. Now again we can conclude that v_2 can't be distinguished, because otherwise v_1 contains an uncharged distinguished variable, and is either p or an element of $\mathcal{D}$.

We have shown that all vertices of V_2 along a path from t to s in G are nondistinguished variables, and each is uncharged. Therefore any V_1 vertex on such a path is forced to be either p, or an element of $\mathcal{D}$. Since in particular this holds for the last vertex (i.e., s) on the path, we conclude that u itself must be an element of $\mathcal{D}$, which is a contradiction because the sets $\mathcal{D}$ and $\mathcal{U}$ are disjoint.

So no path from s to t exists in G', as claimed.

∎

Therefore, to solve an instance of the rule irreducibility problem of size n, we may consider all possible "starter pairs" $u = e_i \in \mathcal{U}$ and $d = e_j \in \mathcal{D}$ in turn. For each starter pair we solve the corresponding bipartite disconnect set problem to determine if the given starter pair can be completed to form an actual partition of $\mathcal{E} \setminus p$ meeting the desired cost bound. There are only $O(n^2)$ possible starter pairs to be considered, and each such constructed bipartite disconnect set problem has $O(n)$ vertices and $O(n^2)$ edges. We shall therefore have a polynomial time algorithm for rule irreducibility if the bipartite disconnect set problem can be solved in polynomial time.

We shall transform a given instance $G = (V_1 \cup V_2, E)$ of bipartite disconnect set into a flow problem in directed graphs. An edge-weighted directed graph D with new source node s_0 and sink node t_0 is obtained from G as follows. First, we replace each node $v \in V_2$ in G by a directed edge (v^l, v^r) between new nodes v^l and v^r of D. The capacity of each such created edge is made to be equal to one. Then, for each edge (w, v) of G with $w \in V_1$ and $v \in V_2$, we replace this edge by two directed edges (w, v^l) and (v^r, w). Each of these edges is made to have capacity infinity. Finally, a new sink node s_0 and a source node t_0 are created in D, and two edges of infinite capacity (s_0, s) and (t, t_0) are added to D.

In Figure 5.2 we illustrate the construction of the graph D from a particular instance G of bipartite disconnect set.

Suppose we use a known polynomial time algorithm, for example Dinits's algorithm [Dini70], to find a maximal flow F from s_0 to t_0 in the network D. Then excepting the

edges (s_0, s) and (t, t_0) in D, the flow F must be a 0-1 flow (i.e., every edge must have a flow of either 0 or 1 along it). It is our claim that the cardinality C of a minimal bipartite disconnect set $C \subseteq V_2$ in G is equal to the cardinality M of a minimum cut in the flow network D.

To prove the claim, we shall show that the inequalities

$$M \leq |C| \leq F$$

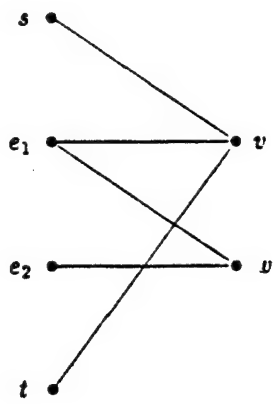
are valid. By the max-flow min-cut theorem, we also know that $M = F$, so that we shall be able to conclude that $M = |C|$, as claimed.

First, we need to show that $M \leq |C|$. In proof, suppose that s can be disconnected from t in G by the removal of the vertices $C \subseteq V_2$. Then a $(W, \dot{W})$ cut of the network D with capacity $|C|$ can be constructed from C in the following way. (Recall that a *cut* in a flow network D is a partition $(W, \dot{W})$ of its vertices such that the source node s_0 is in W, and the sink node t_0 is in $\dot{W}$; its *capacity* is the sum of the capacities of all arcs that cross the cut in the "forward direction"—see, for example [PaSt82]). First, we put s_0 in W, and we put t_0 in $\dot{W}$. Then, for each node $v \in C$ in G, we put the corresponding node v^l into W, and we put the node v^r into $\dot{W}$. All remaining nodes of D are put into W. The result is a $(W, \dot{W})$ cut of the network D, and its capacity is precisely $|C|$. Therefore a *minimal* cut necessarily has its capacity upper bounded by $|C|$, as claimed.

Next, we need to show that $|C| \leq F$. Here, it suffices to show how a maximal flow in the network D yields a disconnect set $C \subseteq V_2$ in the corresponding bipartite disconnect set problem. The construction is simple. Given a maximal flow in D, we look at each edge pair (v^l, v^r), and see if it has 0 or 1 flow along it. If the flow is 1, we remove the corresponding node $v \in V_2$ from G, and if zero, then the corresponding node is not removed in G. The given nodes must disconnect s from t in G, for otherwise the given D flow can be augmented, contradicting its maximality.

Dinits's algorithm is known to run in time $O(|V||A|^2)$ on networks with $|V|$ nodes and $|A|$ arcs. The entire flow graph algorithm therefore has complexity $O(n^2 \times n \times (n^2)^2) = O(n^7)$, as claimed.

Dinits's algorithm is known to run in time $O(|V||A|^2)$ on networks with $|V|$ nodes and $|A|$ arcs. The entire flow graph algorithm therefore has complexity $O(n^2 \times n \times (n^2)^2) = O(n^7)$.



An instance G of bipartite disconnect set...

...becomes a flow problem D.

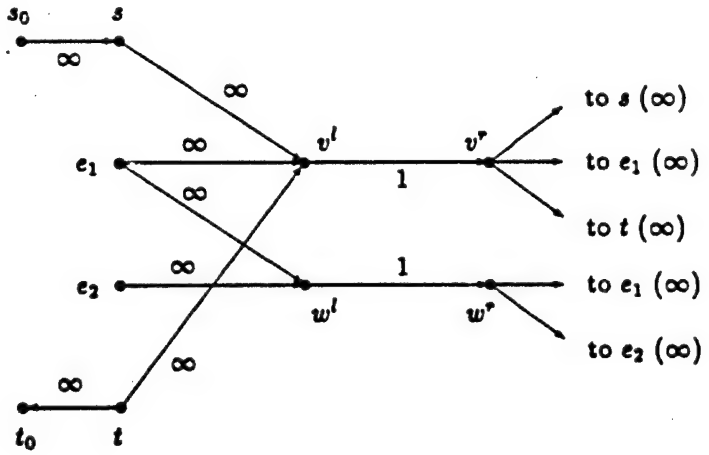


Figure 5.2: Bipartite disconnect set as a flow problem

5.5 Unique factorization

When does a rule admit a *unique* factorization into irreducible rules? Unfortunately, the answer comes back fairly quickly—not very often. There are at least two major difficulties:

1. **Rule (non)commutativity:** Because general rules do not commute, we must think of the two general rule products uv and vu as "essentially different." Yet when u and v do happen to satisfy $uv = vu$, then uv and vu represent the same rule. Any general unique factorization theorem would have to handle this difficulty in some way.

2. **Freedom of choice:** When at least one factorization $r = r_1 r_2$ exists for a rule r, we often shall have several choices available to us between different selections of charged variables in the recursive subgoal of r_1. The different selections and placements of these variables will then lead to other factorizations $r = s_1 s_2$ where $s_1 \neq r_1$ and $s_2 \neq r_2$. For example, applying the Γ-multigraph algorithm to the same generation rule

$$r : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(AB)} \ \& \ \mathtt{e(X_1A)} \ \& \ \mathtt{e(X_2B)},$$

one may obtain the factorization $r = s_1 s_2$ with the rules

$$s_1 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(AX_1)} \ \& \ \mathtt{e(X_2A)}$$
$$s_2 : \mathtt{p(X_1X_2)} \ :- \ \mathtt{p(X_1A)} \ \& \ \mathtt{e(X_2A)},$$

as well as the factorization $r = r_1 r_2$ from Example 5.6. The charged variable set corresponding to both factorizations is $\{X_1, A\}$, and the essential difference between them is the order in which these two variables occur in the top-level recursive subgoals r_1 and s_1.

Amongst the multiple factorizations of a rule we can sometimes discover curious ways to write simple logic programs. For an example, suppose we set ourselves the task of writing a logic program Π to compute a "same generation or one older" relation p from a given "parent" EDB relation $par(X, Y)$. More precisely, we shall require the program Π to compute all tuples $p(X, Y)$ where either X and Y have a common parent ancestor at the same parent depth $d \geq 0$, or X and Y have a common ancestor a at the depth $d + 1$ from X and at the depth d from Y.

A first crack at writing it might yield the following program.

$$
\begin{aligned}
r_1 &: \mathtt{sg}(\mathtt{X}_1\mathtt{X}_1) \\
r_2 &: \mathtt{sg}(\mathtt{X}_1\mathtt{X}_2) \ :- \ \mathtt{sg}(\mathtt{A},\mathtt{B}) \ \&\ \mathtt{par}(\mathtt{X}_1,\mathtt{A}) \ \&\ \mathtt{par}(\mathtt{X}_2,\mathtt{B}). \\
r_3 &: \mathtt{p}(\mathtt{X}_1,\mathtt{X}_2) \ :- \ \mathtt{sg}(\mathtt{X}_1,\mathtt{X}_2). \\
r_4 &: \mathtt{p}(\mathtt{X}_1,\mathtt{X}_2) \ :- \ \mathtt{sg}(\mathtt{X}_2,\mathtt{A}) \ \&\ \mathtt{par}(\mathtt{X}_1,\mathtt{A}).
\end{aligned}
$$

Here, we can think of the program Π as first computing all the same generation facts sg by using a basis rule r_1 and the standard same generation rule r_2. Next, Π computes the relation p from sg by including all the sg facts (rule r_3), and the "off by one generation" facts are thrown in too (rule r_4).

However, the interesting thing is that the same generation rule admits a factorization $sg = s^2$, where s is the *one-half same generation rule*

$$s : \mathtt{sg}(\mathtt{X}_1,\mathtt{X}_2) \ :- \ \mathtt{sg}(\mathtt{X}_2,\mathtt{A}) \ \&\ \mathtt{par}(\mathtt{X}_1,\mathtt{A}).$$

There is a conspicuous similarity between the rule body of s and that of the rule r_4 in Π. In fact, the entire program Π may be replaced by the program $\Pi' = \{r, s\}$

$$
\begin{aligned}
r &: \mathtt{p}(\mathtt{X}_1,\mathtt{X}_1) \\
s &: \mathtt{p}(\mathtt{X}_1,\mathtt{X}_2) \ :- \ \mathtt{p}(\mathtt{X}_2,\mathtt{A}) \ \&\ \mathtt{par}(\mathtt{X}_1,\mathtt{A}),
\end{aligned}
$$

which computes the desired relation p directly. Here, the standard same generation facts will be computed by the even-power recursive expansions of s, while the "off by one" facts will be picked up by the odd powers of s.

5.6 Graph semigroups

But the last example is probably best regarded as a curiousity—in general, the absence of unique factorization properties for rule sets proves to be more a hindrance than a help. However, there is a natural context in which semigroups and unique factorization properties can be exploited to study the query containment properties of a rule set. We shall take Ioannidis's [Ioan89] *partial commutativity problem* as our starting point.

Ioannidis [Ioan89] has drawn attention to the problem of finding "ways to take advantage of partial commutativity, i.e., when the transitive closure of a product of operators is to be

computed, only a subset of which are mutually commutative." To restate the problem in our language, we may generally ask what can be said about the query containment properties of rule sets $\mathcal{S} = \{r_1, \ldots, r_k\}$ for which we may know several algebraic relationships of the form $r_i r_j \simeq r_j r_i$. Such relations may be known to us either by using the techniques outlined above, or from the results of other authors [Ioan89], [RSUV89].

In the semigroup world, such a set of abstract relations is said to be a *presentation of a graph semigroup* [KMR82], and useful results from this literature may be carried over to the study of the partial commutativity problem for rule sets.

Let $\mathcal{S} = \{r_1, \ldots, r_k\}$ be a rule set, and let $w \simeq r_{i_1} r_{i_2} \cdots r_{i_s}$ be an arbitrary recursive expansion of the rules in $\mathcal{S}$. If $\mathcal{S}$ is strongly free, then we know that w determines the rules $r_{i_1}, r_{i_2}, \cdots, r_{i_s}$ uniquely—that is, if $w \simeq r_{j_1} r_{j_2} \cdots r_{j_t}$ then $s = t$ and $i_l = j_l$ for $1 \leq l \leq s$ (Theorem 3.8). Therefore it makes sense to say that w (and indeed every recursive expansion of the rules in $\mathcal{S}$) has a *unique factorization* over $\mathcal{S}$.

However, now suppose to the contrary that $\mathcal{S}$ is not strongly free, but instead satisfies some commutative algebraic relationships of the form $r_i r_j \simeq r_j r_i$. Then we can no longer speak directly about unique factorizations—for example, $w \simeq r_1 r_2 r_3 \simeq r_2 r_1 r_3$ if r_1 and r_2 commute. However, we can prove unique factorization result of a slightly different kind. We need a preliminary definition.

Again, let $\mathcal{S} = \{r_1, \ldots, r_k\}$ be a rule set, and let

$$w \simeq w_1 w_2 \cdots w_l \tag{5.1}$$

be a recursive expansion of the rules in $\mathcal{S}$ where each w_i is either a rule of $\mathcal{S}$ or alternatively w_i is itself a recursive expansion of rules in $\mathcal{S}$. If for each pair w_i, w_j with $1 \leq i, j \leq l$ we find that $w_i w_j \simeq w_j w_i$, then we call the expression 5.1 a *factorization (of w) into commuting parts.*

It is our present goal to prove:

Theorem 5.11 *Suppose $\mathcal{S} = \{r_1, \ldots, r_k\}$ is a rule set, and suppose that every nontrivial algebraic relationship satisfied by the rules of $\mathcal{S}$ is a consequence of relations of the form $r_i r_j \simeq r_j r_i$. Then (i) there exists a unique factorization of any recursive expansion $w \in \mathcal{J}(\mathcal{S})$ into commuting parts, and (ii) we may decide membership for the set of all elements of $\mathcal{J}(\mathcal{S})$ that commute with a particular expansion w in polynomial time.*

Proof Treating the r_i's as formal symbols, define a graph Γ on $\mathcal{S}$ by taking (r_i, r_j) as an edge of Γ if and only if r_i and r_j do *not* commute. Let G be the rule expansion semigroup

$\mathcal{J}(S)$, and take $w \in G$ as a typical formal word over the alphabet S. Then we shall use $C(w)$ to denote the *centralizer*

$$C(w) = \{w_0 \mid w_0 w \simeq w w_0\}$$

of w, and we shall use $\Gamma(w)$ to refer to the *full subgraph*

$$\Gamma(w) = \{(r_i, r_j) \in \Gamma \mid r_i, r_j \text{ occur in } w\}$$

generated by the vertices r_i which occur in w. Let the connected components of $\Gamma(w)$ be

$$\Gamma_1(w), \ldots, \Gamma_s(w),$$

and let w_i, the *ith commutative projection* of w, denote the product of the vertices obtained from w by deleting all vertices not in $\Gamma_i(w)$. Then [KMR82] w_i is well-defined, $w_i w_j \simeq w_j w_i$ for all i, j and

$$w \simeq \prod_{i=1}^{s} w_i.$$

Finally, we define $\pi(w) = \{w_1, \ldots, w_s\}$ as the set of all commutative projections of w. (The present notations should not be confused with those used in Section 4, above).

An element $w \in G$ is called a *c-prime* if w cannot be expressed as a nontrivial product $w \simeq uv \simeq vu$ of two commuting factors $u, v \in G$. A *c-factorization* of w is an equation $w \simeq y_1 \ldots y_k$ where for each i and j we have $y_i y_j \simeq y_j y_i$. We can now translate the central result from [KMR82] into the present context. Because

Every element $w \in \mathcal{J}(S)$ has a unique c-factorization into c-primes,

we have a proof of part (*i*) of the theorem statement. In fact, it is possible to give a simple characterization of the c-primes that occur in the c-factorization of the element w. Let $w \in G$. Then there exists an element $\text{root}(w) \in G$ such that $(\text{root}(w))^m = w$ for some $m > 0$, and for any $x \in G$ such that $x^n = w$ for some $n > 0$, there exists $a > 0$ such that $\text{root}(w)^a = x$. (For a proof, again consult [KMR82]). One shows then that c-primes which occur in the c-factorization of the element w are roots of the commutative projections $w_i \in \pi(w)$.

Finally, we come to the much simpler part (*ii*) of the theorem statement. Here, it is perhaps not surprising that to test the commutativity of two elements w and v, it suffices

to attempt to transform wv into vw by a sequence of interchanges of commuting vertices r_i and r_j. A simple greedy quadratic time algorithm suffices, although we shall present a more sophisticated (linear time) algorithm in Section 6.2 below. ∎

Chapter 6 in part presents a case study of how Theorem 5.11 may be applied to the study of a commonly encountered class of rule sets, the *linear chain rules*. Still, it is useful to pause at this point to give an example of a particular rule set to which Theorem 5.11 applies.

Example 5.12 Consider the three rules

$$r : \mathrm{p}(X_1X_2X_3) \ :- \ \mathrm{p}(X_1X_2A) \ \& \ \mathrm{e}(AX_3)$$
$$s : \mathrm{p}(X_1X_2X_3) \ :- \ \mathrm{p}(X_1X_2A) \ \& \ \mathrm{f}(AX_3)$$
$$t : \mathrm{p}(X_1X_2X_3) \ :- \ \mathrm{p}(AX_2X_3) \ \& \ \mathrm{g}(AX_1)$$

Here, we find that the rules r and s do not commute, while both s and t, and r and t, do commute. The graph Γ is therefore

and it has two connected components. Every recursive expansion w of the rules $\{r, s, t\}$ will therefore have a unique factorization into at most two commuting parts that are obtained from w by commutative projection onto the components of Γ; for example, the expansion $w = trrstr$ can be factored as $w \simeq w_1w_2$, where $w_1 = rrsr$ and $w_2 = tt$.

∎

Finally, it is interesting to note that the centralizer $C(w)$ of a particular element w in a graph semigroup G is a subsemigroup of G that is again a graph semigroup (that is, it is presented by relations of the form $r_ir_j = r_jr_i$) [KMR82]. Thus it appears that to optimize such programs using rule commutativity properties unavoidably involves graph semigroup analysis at some level.

Chapter 6

A case study: linear chain rules

Ullman and Van Gelder [UlVa85] define an interesting class of rule sets whose query containment properties can be studied by using a combination of factorization techniques and graph semigroup analysis. These are the rule sets whose elements are *linear chain rules*. We need a preliminary definition.

An *elementary chain rule* is a rule of the form

$$p(X_1X_2) \ :- \ r_1(X_1, A_1) \ \& \ r_2(A_1, A_2) \ \& \ \cdots \ \& \ r_k(A_{k-1}, X_2)$$

where all predicates are binary and $X_1, A_1, \ldots, A_{k-1}, X_2$ are all distinct variables [UlVa85]. In a *linear elementary chain rule*, exactly one of the predicate names r_i is p.

Example 6.1 The rule

$$p(X_1X_2) \ :- \ q_1(X_1, A_1) \ \& \ p(A_1, A_2) \ \& \ q_2(A_2, X_2)$$

is a linear elementary chain rule. ∎

Ullman and Van Gelder focus primarily on elementary chain rules with nonlinear recursion allowed, and they show how the recursive expansions of such rules can be simply modelled by context-free grammars. The latter observation is then used to demonstrate that the computation of the minimum model of such programs is efficiently parallelizable (i.e., in $\mathcal{NC}$) when a certain *polynomial fringe property* is satisfied.

In the linear case, that the minimum model computation is in $\mathcal{NC}$ is immediate because such rules are "thinly disguised transitive closure" rules [UlVa85] for which minimum models may be computed by *path doubling* techniques.

Suppose $\mathcal{S}$ is a rule set whose elements are all elementary chain rules, here momentarily with nonlinear recursion allowed (i.e., multiple $\mathtt{r}_i$'s may be p). If one attempts to use the context free grammar approach to study query containments amongst recursive expansions of the rules in $\mathcal{S}$, then discovering nontrivial containments is as hard as deciding the *ambiguity problem* for context free grammars, which is known to be undecidable.

However, we shall see that *linear* elementary chain rules have particularly simple factorization properties, even when we widen our perspective to generalized (i.e. nonelementary) forms of such rules. These factorization properties, together with graph semigroup techniques, will allow us to present nontrivial query containment algorithms for such rules. We need another definition.

Definition 6.2 *A* linear chain rule *is a rule of the form*

$$\mathtt{p} \ :- \ \mathtt{r}_1 \ \& \ \mathtt{r}_2 \ \& \ \cdots \ \& \ \mathtt{r}_k$$

where the arities of the p *and* $\mathtt{r}_i$ *predicates may be arbitrary, and the following conditions are satisfied:*

- (CHAIN CONDITION) *For* $1 \leq i < j \leq k$, *the predicates* $\mathtt{r}_i$ *and* $\mathtt{r}_j$ *share a nondistinguished variable if* $\mathtt{j} = \mathtt{i} + 1$, *and otherwise share none.*

- (L/R BLOCK CONDITION) *Each distinguished variable occurs exactly once in the body, either in* $\mathtt{r}_1$, *or alternatively in* $\mathtt{r}_k$. *Moreover* r_1 *and* r_k *each have at least one distinguished variable occurring in them.*

- (LINEAR CONFORMITY) *Exactly one* $\mathtt{r}_i$ *is* p. *Moreover, if* $\mathtt{r}_i = \mathtt{p}$ *has the same variable occurring at positions* α *and* β, *then the distinguished variables* X_α *and* X_β *must either both occur in* $\mathtt{r}_1$, *or alternatively both occur in* $\mathtt{r}_k$.

Example 6.3 Of the many linear recursive rules we have considered in this thesis, only a few have not been linear chain rules. Here are some of the linear chain rules we have seen so far:

The transitive closure rule for directed graphs:

$$\mathtt{p(XY)} \ :- \ \mathtt{p(XA)} \ \& \ \mathtt{e(AY)}.$$

The same generation rule:

$$p(X_1X_2) \ :- \ e(X_1A) \ \& \ p(AB) \ \& \ e(X_2B)$$

The one-half same generation rule:

$$sg(X_1, X_2) \ :- \ sg(X_2, A) \ \& \ par(X_1, A).$$

Left-right transitive closure rules:

$$p(X_1X_2) \ :- \ p(X_1C) \ \& \ e(CB) \ \& \ e(AB) \ \& \ e(AX_2)$$
$$p(X_1X_2) \ :- \ p(X_1B) \ \& \ e(BA) \ \& \ e(X_2A)$$

Many more rules meet the definition. For example:

$$p(X_1X_2X_3) \ :- \ p(X_1CX_3) \ \& \ e(CBBD) \ \& \ f(DBAB) \ \& \ g(AX_2) \qquad (6.1)$$
$$p(X_1X_2X_3X_4) \ :- \ e(X_1A) \ \& \ f(ABCD) \ \& \ p(DEEE) \ \& \ f(X_4X_2EX_3) \qquad (6.2)$$

For some examples of rules that do not quite meet the definition, we can take the three rules

$$p(X_1X_2X_3) \ :- \ p(X_1CX_3) \ \& \ e(CBB) \ \& \ f(BAB) \ \& \ f(A\underline{C}X_2)$$
$$p(X_1X_2X_3X_4) \ :- \ e(X_1A) \ \& \ f(A\underline{X_2}CD) \ \& \ p(DEEE) \ \& \ f(X_4X_2EX_3)$$
$$p(X_1X_2X_3X_4) \ :- \ e(X_1A) \ \& \ f(ABCD) \ \& \ p(DEE\underline{D}) \ \& \ f(X_4X_2EX_3)$$

which violate the chain condition, the block condition, and the linear conformity condition, respectively, from top to bottom. We have underlined the violation in each of the three clauses. ∎

Here, our interest will be restricted to linear chain rules that satisfy a stronger version of the chain condition:

- (STRONG CHAIN CONDITION) For $1 \leq i < j \leq k$, the predicates r_i and r_j share a *unique* nondistinguished variable *if and only if* $j = i + 1$.

We impose the strong chain condition rather than the normal chain condition only in order to ensure that the linear chain rules we consider will have convenient factorization properties. For the remainder of this chapter, *we shall assume all linear chain rules to satisfy the strong chain condition* (but we remind the reader that our class is more restrictive than the original definition [UVa85] specifies). However, note that the classes of rules mentioned above do meet the strong chain condition (although rules 6.1 and 6.2 do not).

Suppose we are given a set of linear chain rules $S = \{r_1, \ldots r_c\}$ defining a relation p, and we are considering two finite-depth recursive expansions $r_a r_b \cdots$ and $r_s r_t \cdots$ of the rules S. How can we efficiently test whether $r_a r_b \cdots \geq r_s r_t \cdots$? Our goal in the present chapter to give two algorithms for such query containment problems. Our methods will not be guaranteed to capture all the query containments such a rule set S may satisfy, however. Still, much nontrivial information can be deduced.

Of the two algorithms we present, the first will be the simpler, and will depend only on the unique factorization properties present in graph semigroups. However, the first method will has the disadvantage of not exploiting all the commutativity information that is present for general linear chain rules.

The second algorithm will use a more advanced technique due to P. Cartier and D. Foata [CaFo69], and in a strong sense can be said to exploit "everything we know about commutativity" for linear chain rules in the study of query containments.

The following 3 observations will be important to our methods.

1. Linear chain rules have a particularly simple factorization structure. In fact, every linear chain rule with k EDB subgoals can always be efficiently and completely factored into a product of k irreducible rules with 1 EDB subgoal apiece.

2. There is a strong commutativity test for rules with 1 EDB subgoal [RSUV89].

3. In light of points 1 and 2, the graph semigroup method from Chapter 5 may be brought to bear on the study of query containments.

6.1 Query containment by commutative decomposition

Our first method for query containment testing is most easily illustrated with an extended example.

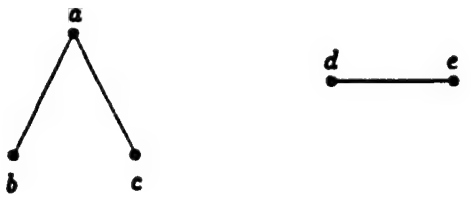
Example 6.4 We shall take the rule set $S = \{r_{ab}, r_c, r_d, r_e\}$ consisting of the four linear chain rules

$$\begin{aligned} r_{ab} &: \mathtt{p(X_1X_2X_3X_4)} \ :- \ \mathtt{p(X_1AX_3B)} \ \& \ \mathtt{e(CA)} \ \& \ \mathtt{f(CX_2X_4)} \\ r_c &: \mathtt{p(X_1X_2X_3X_4)} \ :- \ \mathtt{p(AX_2X_3B)} \ \& \ \mathtt{f(X_1AX_4)} \\ r_d &: \mathtt{p(X_1X_2X_3X_4)} \ :- \ \mathtt{p(X_1X_2AX_4)} \ \& \ \mathtt{e(AX_3)} \\ r_e &: \mathtt{p(X_1X_2X_3X_4)} \ :- \ \mathtt{p(X_1X_2AX_4)} \ \& \ \mathtt{e(X_3A)} \end{aligned}$$

as moderately complex running example. First, one quickly verifies that the strong chain, block, and linear conformity conditions are satisfied for each of these three rules. The notation we have chosen the first rule, r_{ab}, stems from the fact that it has a complete factorization into single EDB subgoal rules $r_{ab} = r_a r_b$, where the latter two rules are given by

$$\begin{aligned} r_a &: \mathtt{p(X_1X_2X_3X_4)} \ :- \ \mathtt{p(X_1AX_3B)} \ \& \ \mathtt{f(AX_2X_4)} \\ r_b &: \mathtt{p(X_1X_2X_3X_4)} \ :- \ \mathtt{p(X_1AX_3X_4)} \ \& \ \mathtt{e(X_2A)}. \end{aligned}$$

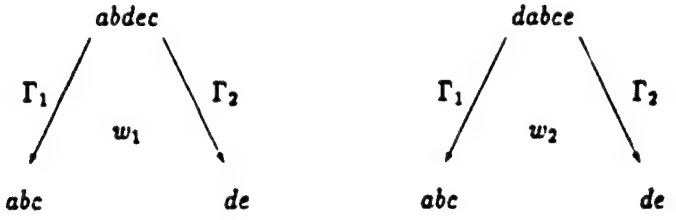
We shall see below that such complete factorizations always exist for linear chain rules, and that they may be found efficiently. The mutual commutativity relationships $r_i r_j \simeq r_j r_i$ for the five rules $\{r_a, r_b, r_c, r_d, r_e\}$ can be summarized by a graph Γ



where (in a manner similar to Theorem 5.11) we have drawn edges between vertices that do *not* commute. Again, we shall see below that this commutativity information may be efficiently computed in the general case by applying a criterion due to Ramakrishnan, Sagiv, Ullman, and Vardi [RSUV89]. The graph Γ has connected components $\Gamma_1 = \{a, b, c\}$ and $\Gamma_2 = \{d, e\}$.

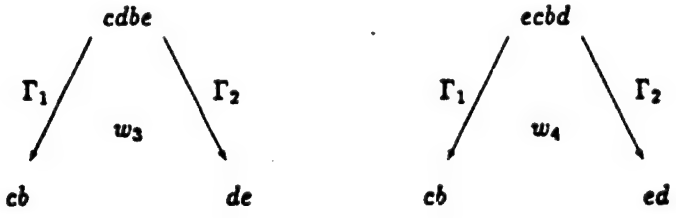
Now, suppose for the sake of an example that we would like to test whether the query containment $w_1 \succeq w_2$ holds between the recursive expansions $w_1 = r_{ab} r_d r_e r_c$ and $w_2 =$

$r_d r_{ab} r_c r_e$ of the rules S. Still following the graph semigroup method from Theorem 5.11, we begin by calculating the commutative projections



of the words w_1 and w_2 onto the components Γ_1 and Γ_2. Because we find that w_1 and w_2 have identical factorizations into the commuting factors $r = r_a r_b r_c$ and $s = r_d r_e$, we conclude that the query containment $w_1 \succeq w_2$ does indeed hold amongst the recursive expansions of S.

To take a second example, suppose we are interested in testing the query containment $w_3 \succeq w_4$, where w_3 and w_4 are the recursive expansions $w_3 = r_c r_d r_b r_e$ and $w_4 = r_e r_c r_b r_d$. Here the respective commutative projections are



and because the projections onto Γ_2 differ, (we have $de \neq ed$), we cannot conclude that the given query containment holds (and in fact it does not).

∎

To complete our description of chain rule query containment testing by commutative decomposition, we need to check the the points left incomplete above. First, we claimed above that

Lemma 6.5 *Every linear chain rule r with k EDB subgoals can be factored completely as a recursive expansion of k rules, each of which has 1 EDB subgoal apiece.*

Proof Suppose the rule r has recursive arity n and takes the form

$$r : p \ :- \ r_1 \ \& \ r_2 \ \& \ \cdots \ \& \ r_k,$$

where we shall suppose that it is the subgoal r_j in the body of r that corresponds to the occurrence of the recursive p subgoal. By renaming nondistinguished variables if necessary, we may also assume that the nondistinguished variable shared by r_i and r_{i+1} is named A_i for each value i with $1 \leq i < k$.

We can prove the lemma directly by presenting a k level abstract expansion tree for r, where at each level exactly one EDB subgoal of r appears. The structure of the tree is as suggested by Figure 6.1.

Here, the variable patterns in the leaf nodes $r_1, r_2, \ldots, r_k$ correspond exactly to their counterparts in the rule r, and all we have to do is demonstrate how variables may be passed down through the intermediate p-subgoals $p_1, p_2, \ldots, p_{j-1}, p_k, p_{k-1}, \ldots, p_{j+1}$ so as to guarantee that the resulting tree does indeed correspond to an actual recursive expansion of rules.

We consider the intermediate p-subgoals in turn, starting with p_1. Clearly we must take p_1's variables as $X_1, X_2, \ldots, X_n$ in order, because this is forced on us at the root node of all expansion trees. Next, for indices i in the range $2 \leq i \leq j-1$, we must first take p_i to contain the nondistinguished variable A_i that is to be shared between r_i and r_{i+1}. Also, every distinguished variable that is to occur in r_k must also appear somewhere in p_i. Because by definition r_k can contain at most $n-1$ different distinguished variables, there are at most n variables (nondistinguished and distinguished) all told that must appear in p_i, and there is room for all these variables amongst the n argument positions of p_i.

Next, we turn to p_k. This subgoal must contain A_{k-1} together with whatever distinguished variables are to appear in r_k. Again there are at most n such variables, so they all fit inside p_k.

Finally, for the indices i in the range $k-1 \geq i \geq j$, the subgoal p_i must contain the two variables A_i and A_{j-1}, the nondistinguished variable that is to be shared between r_{j-1} and r_j. No distinguished variables need appear, because all such variables have already occurred in r_1 and r_k, above p_i in the expansion tree.

∎

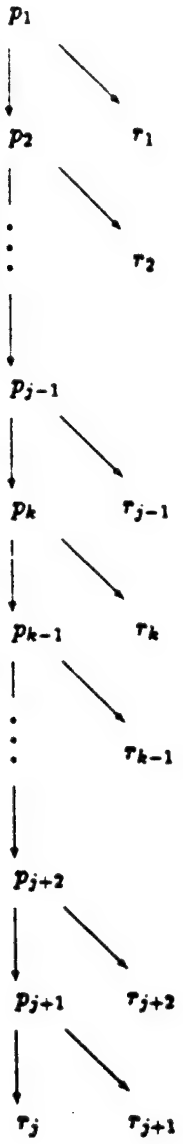


Figure 6.1: An expansion tree for a chain rule with $r_j = p$.

Secondly, there is also the matter of testing for commutativity. In [RSUV89], the following simple condition is presented.

Theorem 6.6 *Consider two safe, Datalog linear rules of the form*

$$r_1 : \mathtt{p}(\mathtt{X}_1,\ldots,\mathtt{X}_n) \ :- \ \mathtt{p}(\mathtt{Y}_1,\ldots,\mathtt{Y}_n) \ \&\ \mathtt{G}_1 \ \&\ \cdots \ \&\ \mathtt{G}_s,$$
$$r_2 : \mathtt{p}(\mathtt{X}_1,\ldots,\mathtt{X}_n) \ :- \ \mathtt{p}(\mathtt{Z}_1,\ldots,\mathtt{Z}_n) \ \&\ \mathtt{H}_1 \ \&\ \cdots \ \&\ \mathtt{H}_t$$

Then the following conditions are sufficient for the commutativity law $r_1 r_2 = r_2 r_1$:

1. *If $Y_i \neq X_i$ then X_i does not appear among the H's.*
2. *If $Z_i \neq X_i$ then X_i does not appear among the G's.*
3. *If $Y_i = X_j$ for some $j \neq i$, then $Z_i = X_i$ and $Z_j = X_j$.*
4. *If $Z_i = X_j$ for some $j \neq i$, then $Y_i = X_i$ and $Y_j = X_j$.*

The theorem gives us a simple method to test for commutativity between the constituent factors of a set of linear chain rules. It should be observed, however, that because we are always testing for commutativity between two single EDB rules, there is no real loss of efficiency in the alternative of simply expanding the two rules r_1 and r_2 in both orders and testing all possible query containment mappings between them exhaustively.

6.2 The V-decomposition

The commutative decomposition method fails to recognize some simple query containments that are consequences of partial commutativity. For example, in Example 6.4 the decomposition method would fail to discover that $r_a r_b r_c \succeq r_a r_c r_b$. Even though this containment follows immediately from the relationship $r_b r_c \simeq r_c r_b$, the projections onto the connected component Γ_1 differ (and in fact are *abc* in the first case, and *acb* in the second).

In order to capture all the consequences of partial commutativity we may apply a second idea called the ***V*-decomposition**, first described by Cartier and Foata [CaFo69].

Suppose that $S = \{r_a, r_b, \ldots\}$ is a finite rule set whose mutual commutativity relationships $r_i r_j \simeq r_j r_i$ have been summarized by a graph Γ on the vertices $\{a, b, \ldots\}$, where as usual we put edges between vertices that do not commute. We shall say that a query containment $w_1 \succeq w_2$ between recursive expansions w_1 and w_2 of the rules S is a *consequence of partial commutativity* if the word w_1 can be transformed into the word w_2 by a sequence of interchanges of commuting rules $r_i r_j \simeq r_j r_i$.

```
begin
    Create a single cell with the first letter of w in it.
    for i = 2 to length(w) do begin
       (1) Let v be the ith letter in w.
       (2) Find the rightmost cell c that either contains the
           letter v, or contains a symbol u not commuting with
           v.  Then place v in the next cell to the right of
           c, respecting the alphabetical order within this
           cell.  If c is already the rightmost cell, then create
           a new rightmost cell with v only in it.  If c does not
           exist, then place v in the leftmost cell, again
           respecting the alphabetical order within that cell.
       end
    end
```

Figure 6.2: Computing the V-decomposition $V(w)$ of a word w.

We shall need to put an arbitrary (but fixed) strict linear order on the alphabet in order to describe V-decompositions. Here we shall take the convenient alphabetic ordering $a < b < c < \cdots$.

The *V-decomposition* of a word w over the alphabet $\Omega = \{a, b, \ldots\}$ is a symbol $V(w)$ of the form

$$w_1 \mid w_2 \mid \ldots \mid w_k,$$

where the w_i's are also words over the alphabet Ω. The w_i's are called the *cells* of $V(w)$.

To compute $V(w)$ from w, we apply the algorithm in Figure 6.2.

Example 6.7 We shall compute the V-decomposition $V(w)$ of the word $w = abcdaeccead$, where mutual commutativity relationships are summarized by the graph

Following the algorithm, we obtain the following sequence of cell expressions through each pass of the main loop:

$$a$$

$$a \mid b$$

$$a \mid bc$$

$$ad \mid bc$$

$$ad \mid bc \mid a$$

$$ad \mid bce \mid a$$

$$ad \mid bce \mid a \mid c$$

$$ad \mid bce \mid a \mid c \mid c$$

$$ad \mid bce \mid ae \mid c \mid c$$

$$ad \mid bce \mid ae \mid c \mid c \mid a$$

$$ad \mid bce \mid ae \mid cd \mid c \mid a$$

The final expression gives the V-decomposition of the word w. We have

$$V(w) \;=\; ad \mid bce \mid ae \mid cd \mid c \mid a.$$

∎

V-decompositions are relevant to the study of query containments because of the following two results [CaFo69].

1. Ignoring the cell dividers in $V(w)$, we have a single word that is commutatively equivalent to w.
2. Two words w_1 and w_2 are commutatively equivalent if and only if their V decompositions coincide.

Thus, to test whether a given query containment $w_1 \supseteq w_2$ is true as a consequence of partial commutativity, we may simply compute $V(w_1)$ and $V(w_2)$, and see if they are identical.

Thus, in a strong sense the V-decomposition captures "everything that we know about query containments from partial commutativity."

We can sum up our Chapter 6 results in the following way. We have given two methods for recognizing query containments between the recursive expansions of linear chain rules, one based on commutative projection, and the other based on the V-decomposition. In the particular case where all the algebraic relationships of a set of linear chain rules are consequences of partially commutative relationships $r_i r_j \simeq r_j r_i$, we have shown how any desired algebraic relationship can be tested efficiently, answering a question asked by Ioannidis. However, the theory is moderately complex and it is not at all clear how an all-purpose algorithm for linear chain rule evaluation could be devised to exploit all the redundancy that can be discovered using our techniques. Certainly *ad hoc* techniques as one finds for example in [IoWo88] can be derived for particular rule sets once some redundancy has been identified, but whether general purpose algorithms capable of exploiting all the known redundancy for general rule sets must remain a topic for future research.

Chapter 7

Conclusion

There are several open problems and possibilities for extension of our work that we would now like to consider briefly.

Can a workable algebraic theory of nonlinear recursive rules be devised? The work of Ioannidis and Wong represents an important first step in this direction. The major difficulties appear to arise with the nonassociate character of nonlinear rule composition, so that the strongest theorems are proved when certain associativity assumptions are made [IoWo88]. Saraiya [Sar89a] discusses conditions for the replacement of nonlinear rules by linear ones via *ZYT linearization*. If a nonlinear rule set can be ZYT linearized, is there an "inherent associativity" in the original nonlinear rule forms that awaits an algebraic characterization?

Is linear rule set freeness decidable? As we remarked above in Chapter 4, the problem is closely related to results on boundedness [Cosm88], [GMSV87], [Ioan85], [NaSa87]. Work on boundedness can seem to be far removed from real life recursions at times, but it is fundamental to our understanding of the redundance and expressibility properties of recursive database queries.

In a sequence of three papers, A. Richard Helm [Hel87], [Hel88a], [Helm88b] has developed a theory of detecting and eliminating redundant derivations in logic programming systems. He uses a control language over derivations to illustrate how under certain circumstances redundant derivations may be made not to occur when a program is executed using his synthesized control expressions. Because his control strategies are essentially regular expressions, it seems likely that redundancy discovered using not only the techniques developed here, but also those of other authors [Dong88], [IoWo88], could be meshed with

Helm's control strategies. We have not studied this idea in the present work, but it does seem to be a natural direction for future research.

Appendix A

Containment depth details

In this appendix we complete the details to our proof of Theorem 4.13, above, checking the six points of the containment mapping definition in turn.

(1). Check that $\phi_1(h^i) = h^{j-p}$.

This is immediate from the definition.

(2). Check that $\phi_1(Q_i(t,\lambda)) \in \mathcal{E}^{j-p}$ for all t and λ.

This is also immediate from the definitions.

(3). Check that if $\sigma_l(h^i)$ is a distinguished variable, then $\sigma_l(\phi_1(h^i)) = \sigma_l(h^{j-p})$ is the same distinguished variable.

Suppose that $\sigma_l(h^i)$ is a distinguished variable. Because ϕ is a containment mapping, $\sigma_l(\phi(h^i)) = \sigma_l(h^j)$ is the same distinguished variable. Next, note that $j > Ki = 4pnki$ implies that

$$j - p > 3pnki > n,$$

i.e., $j - p > n$. We therefore may apply Lemma 4.7, part 1, above, to conclude that $\sigma_l(h^{j-p}) = \sigma_l(h^j) = \sigma_l(h^i)$, as required.

(4). Check that if $\sigma_l(Q_i(t_1,\lambda_1))$ is a distinguished variable, then $\sigma_l(\phi_1(Q_i(t_1,\lambda_1)))$ is the same distinguished variable.

Let $\phi(Q_i(t_1,\lambda_1)) = Q_j(t_2,\lambda_2)$. Two cases arise, according to whether $t_2 > M$ or $t_2 < M - pn + 1$.

Suppose first that $t_2 > M$. We have already noted above that $M \geq n$, and in the

verification of (3), above, we saw that $j - p > n$. Lemma 4.8, point 1, therefore applies, and we conclude $\sigma_l(\mathcal{Q}_i(t_1, \lambda_1)) = \sigma_l(\mathcal{Q}_j(t_2, \lambda_2)) = \sigma_l(\mathcal{Q}_{j-p}(t_2 - p, \lambda_2)) = \sigma_l(\phi_1(\mathcal{Q}_i(t_1, \lambda_1)))$, as required.

On the other hand, suppose $t_2 < M - pn + 1$. Then we can add p to t_2 without making $t_2 \geq j$. Applying Lemma 4.6 a total of p times, we conclude $\sigma_l(\mathcal{Q}_i(t_1, \lambda_1)) = \sigma_l(\mathcal{Q}_j(t_2, \lambda_2)) = \sigma_l(\mathcal{Q}_{j-p}(t_2, \lambda_2))$, as required.

(5). Check that if $\sigma_{l_1}(\mathcal{Q}_i(t_1, \lambda_1)) = \sigma_{l_2}(\mathcal{Q}_i(t_2, \lambda_2))$ then $\sigma_{l_1}(\phi_1(\mathcal{Q}_i(t_1, \lambda_1)))$ is equal to $\sigma_{l_2}(\phi_1(\mathcal{Q}_i(t_2, \lambda_2)))$.

Let $\phi(\mathcal{Q}_i(t_1, \lambda_1)) = \mathcal{Q}_j(s_1, \gamma_1)$, and let $\phi(\mathcal{Q}_i(t_2, \lambda_2)) = \mathcal{Q}_j(s_2, \gamma_2)$.

If $\sigma_{l_1}(\mathcal{Q}_i(t_1, \lambda_1)) = \sigma_{l_2}(\mathcal{Q}_i(t_2, \lambda_2))$ is a distinguished variable, then the verification is carried out as in (4), above. So suppose that $\sigma_{l_1}(\mathcal{Q}_i(t_1, \lambda_1))$ and $\sigma_{l_2}(\mathcal{Q}_i(t_2, \lambda_2))$ are both equal to the nondistinguished variable A^l. Because ϕ is a containment mapping, we know that $v = \sigma_{l_1}(\phi(\mathcal{Q}_i(t_1, \lambda_1)))$ is equal to $\sigma_{l_2}(\phi(\mathcal{Q}_i(t_2, \lambda_2)))$. If the variable v is distinguished, then we may again reduce our argument to that made in (4), above. So we can suppose without loss of generality that v is some nondistinguished variable B^m.

Next, we claim that either both $s_1, s_2 > M$ or both $s_1, s_2 < M - pn + 1$ is the case; i.e., it is impossible that we have, say, $s_1 > M$ while $s_2 < M - pn + 1$. (Recall that by construction, neither s_1 nor s_2 falls in the closed interval $[M - pn + 1, M]$). To prove this, we apply Lemma 4.11. We have $|M + 1 - (M - pn + 1) + 1| = pn + 1 > n$, so that $B^m \in \sigma(\mathcal{Q}_j(s_1, \gamma_1)) \cap \sigma(\mathcal{Q}_j(s_2, \gamma_2))$ implies that either $s_1, s_2 > M$ or $s_1, s_2 < M - pn - 1$.

To finish the argument, then, we must again consider two subcases. In the first subcase we shall assume $s_1, s_2 > M$, and in the second, $s_1, s_2 < M - pn + 1$.

Suppose first that $s_1, s_2 > M$. Because $j - p \geq n$ and $s_1, s_2 > M \geq n$, we may apply Lemma 4.8, part 2 twice to conclude

$$\begin{aligned}
\sigma_{l_1}(\phi_1(\mathcal{Q}_i(t_1, \lambda_1))) &= \sigma_{l_1}(\mathcal{Q}_{j-p}(s_1 - p, \gamma_1)) \\
&= B^{m-p} \\
&= \sigma_{l_2}(\mathcal{Q}_{j-p}(s_2 - p, \gamma_2)) \\
&= \sigma_{l_2}(\phi_1(\mathcal{Q}_i(t_2, \lambda_2))),
\end{aligned}$$

as required.

Finally, suppose that $s_1, s_2 < M - pn + 1$. Then we may apply Lemma 4.6 a total of p

times as in part (4), above, to conclude

$$\sigma_{l_1}(\phi_1(\mathcal{Q}_i(t_1,\lambda_1))) = \sigma_{l_1}(\mathcal{Q}_{j-p}(s_1,\gamma_1)) = B^m = \sigma_{l_2}(\mathcal{Q}_{j-p}(s_2,\gamma_2)) = \sigma_{l_2}(\phi_1(\mathcal{Q}_i(t_2,\lambda_2))),$$

as required.

(6). Check that if $\sigma_{l_1}(\mathcal{Q}_i(t_1,\lambda_1)) = \sigma_{l_2}(h^i)$ then $\sigma_{l_1}(\phi_1(\mathcal{Q}_i(t_1,\lambda_1))) = \sigma_{l_2}(\phi_1(h^i))$.

First, suppose that $\sigma_{l_1}(\mathcal{Q}_i(t_1,\lambda_1)) = \sigma_{l_2}(h^i)$ is a distinguished variable v. Then from part (4), above, we know that $\sigma_{l_1}(\phi_1(\mathcal{Q}_i(t_1,\lambda_1))) = v$; also, by Lemma 4.7, part 1, we have

$$v = \sigma_{l_2}(\phi(h^i)) = \sigma_{l_2}(h^j) = \sigma_{l_2}(h^{j-p}) = \sigma_{l_2}(\phi_1(h^i)),$$

as required.

On the other hand, suppose $\sigma_{l_1}(\mathcal{Q}_i(t_1,\lambda_1)) = \sigma_{l_2}(h^i)$ is a nondistinguished variable, say A^l. Let $\phi(\mathcal{Q}_i(t_1,\lambda_1)) = \mathcal{Q}_j(s_1,\gamma_1)$. If $\sigma_{l_1}(\phi(\mathcal{Q}_i(t_1,\lambda_1))) = \sigma_{l_2}(\phi(h^i))$ is a distinguished variable, then we may apply the argument of the previous paragraph to finish. So suppose $\sigma_{l_1}(\phi(\mathcal{Q}_i(t_1,\lambda_1))) = \sigma_{l_2}(\phi(h^i))$ is a nondistinguished variable B^m. Then $s_1 \geq M$ by Lemma 4.11, and we can conclude as in part (5), above, that

$$\sigma_{l_1}(\phi_1(\mathcal{Q}_i(t_1,\lambda_1))) = \sigma_{l_1}(\mathcal{Q}_{j-p}(s_1-p,\gamma_1)) = B^{m-p} = \sigma_{l_2}(h_{j-p}) = \sigma_{l_2}(\phi_1(h^i)),$$

by an application of Lemma 4.8 and Lemma 4.7.

Bibliography

[Abit89] S. Abiteboul, 1989. Boundedness is undecidable for Datalog programs with a single recursive rule. I.N.R.I.A. unpublished manuscript, 1989.

[AhUl79] A. V. Aho, J. D. Ullman, 1979. Universality of data retrieval languages. *Proc 6th ACM Symp. on Principles of Programming Languages*, 110-117.

[Ban86a] F. Bancilhon, D. Maier, Y. Sagiv, J. D. Ullman, 1986. Magic sets and other strange ways to implement logic programs. In *Proceedings of the ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems*, 1986.

[Banc86b] F. Bancilhon, R. Ramakrishnan, 1986. An amateur's introduction to recursive query processing strategies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington, D.C. 16-52.

[Blu65a] E. K. Blum, 1965. A note on free subsemigroups with two generators, *Bull. Am. Math. Soc.* 71, 678-679.

[Blu65b] E. K. Blum, 1965. Free subsemigroups of a frec semigroup, *Mich. Math. J.* 12, 179-182.

[CaFo69] P. Cartier, D. Foata, 1969. Problemes combinatoires de commutation et rearrangements, Lecture Notes in Mathematics No. 85, Springer-Verlag, Berlin.

[ChHa82] A. K. Chandra, D. Harel, 1982. Structure and complexity of relational queries. *Journal of Computer and System Sciences* 25 (1), 99-128.

[ChHa85] A. K. Chandra, D. Harel, 1985. Horn-clause queries and generalizations. *J. Logic Programming* 1, 1-15.

[ChMe77] A. K. Chandra, Merlin P.M., 1977. Optimal implementation of conjunctive queries in relational databases. *Proc. 9th ACM Symp. on Theory of Computing*, Boulder, 1977, 77-90.

[Cloc81] W. F. Clocksin and C. S. Mellish, 1981. *Programming In Prolog*. Springer-Verlag, New York, New York.

[Codd70] E. F. Codd, 1970. A relational model of data for large shared data banks. *Commun. ACM* **13**, 6 (June), 1970, 377-387.

[Cosm88] S. S. Cosmadakis, H. Gaifman, P. C. Kanellakis, and M. Y. Vardi, 1988. Decidable optimization problems for database logic programs. Proc. 20th ACM Symp. on Theory of Computing, May 1988, 477-490.

[Dini70] E. A. Dinits, 1970. Algorithm for solution of maximal flow in a network with power estimation. *Soviet Math. Dokl.*, **11** (1970) 1277-80.

[Dong88] G. Dong, 1988. On the composition and decomposition of Datalog program mappings. Ph.D. thesis, University of Southern California, December 1988.

[GMN84] H. Gallaire, J. Minker, and J. Nicolas, 1984. Logic and Databases: A Deductive Approach. *ACM Computing Surveys* **16**, 2 (June 1984), 153-185.

[GMSV87] H. Gaifman, H. Mairson, Y. Sagiv, and M. Y. Vardi, 1987. Undecidable optimization problems for database logic programs. *Proc. 2nd IEEE Symp. on Logic in Computer Science*, Ithaca, 1987, 106-115.

[Hel87] A. R. Helm, 1987. Inductive and deductive control of logic programs. *4th International Conference on Logic Programming*, Melbourne, Australia, 488-512.

[Hel88a] A. R. Helm, 1988. Controlling and detecting redundant derivations in logic programming and deductive database systems. Ph.D. thesis, University of Melbourne, Australia, 1988.

[Helm88b] A. R. Helm, 1988. Detecting and eliminating redundant derivations in logic programming systems. Computer Science technical report RC 14243 (#63766), IBM Research Division, T. J. Watson Research Center, Yorktown Heights, New York, November 1988.

[Hu74] T. W. Hungerford, 1974. *Algebra*. Springer-Verlag, New York.

[Ioan85] Y. E. Ioannidis, 1985. Bounded recursion in deductive databases. Technical report UCB/ERL M85/6, UC Berkeley, February 1985.

[Ioan89] Y. E. Ioannidis, 1989. Commutativity and its role in the processing of linear recursion. *Proc. 15th International Conference on Very Large Databases*. Amsterdam, 1989, 155-163.

[IoWo88] Y. E. Ioannidis, and E. Wong, 1988. Towards an Algebraic Theory of Recursion. University of Wisconsin CS technical report No. 801, October, 1988.

[IoRa88] Y. E. Ioannidis and R. Ramakrishnan, 1988. Efficient transitive closure algorithms. Computer Science Technical Report # 765, April 1988.

[JAN87] H. V. Jagadish, R. Agrawal, and L. Ness, 1987. A study of transitive closure as a recursion mechanism, *Proc. 6th Symp. on the Principles of Database Systems*, 1987, 331-344.

[Kane88] P. C. Kanellakis, 1988. Elements of Relational Database Theory. In *Handbook of Computer Science*, North-Holland, (to appear).

[Lall79] G. Lallement, 1979. Semigroups and combinatorial applications. Wiley Interscience, New York.

[Lehm76] D. J. Lehmann, 1976. Algebraic structures for transitive closure. Theory of Computation Report No. 10, Department of Computer Science, University of Warwick, February 1976.

[KMR82] K. H. Kim, L. G. Makar-Limanov, and F. W. Roush, 1982. Graph monoids. *Semigroup Forum* **25**, 1-7.

[Mark62] Al. A. Markov, 1962. Non-recurrent coding. *Problem, Kybern.* **8**, 169-189 (Russian).

[Naug86b] J. Naughton, 1986. Redundancy in function free recursive inference rules. In *Proceedings of the IEEE Symposium on Logic Programming*, 1986.

[Naug86a] J. Naughton, 1986. Data independent recursion in deductive databases, *Proc. 5th Symp. on the Principles of Database Systems*, Cambridge, Massachusetts, March 1986, 267-279.

[NaSa87] J. Naughton and Y. Sagiv, 1987. A decidable class of bounded recursions. In *Proc. 4th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, 227-236.

[Naug87] J. Naughton, 1987. Optimization of recursive database query languages. Ph.D. thesis, Stanford University, May 1987.

[NaSa88] J. Naughton and Y. Sagiv, 1988. Minimizing expansions of recursions. Technical report CS-TR-150-88, Princeton University, April 1988.

[Naug88] J. Naughton, 1988. Compiling separable recursions. *ACM SIGMOD International Conference on Management of Data*, 312-319.

[Ness86] L. Ness, 1986. Reducing linear recursion to transitive closure. Technical report TR-86-25, University of Texas at Austin, November 1986.

[PaSt82] C. H. Papadimitriou, K. Stieglitz, 1982. *Combinatorial Optimization*, Prentice-Hall, Englewood Cliffs, New Jersey.

[Plam89] T. Plambeck, 1989. Containment depth and uniform boundedness of linear recursive single rule Datalog programs, Manuscript submitted for publication.

[Plam90] T. Plambeck, 1990. Semigroup techniques in recursive query optimization, *Proc. 9th Symp. on the Principles of Database Systems*, Nashville, 1990.

[Post47] E. L. Post, 1947. Recursive unsolvability of a problem of Thue, *J. Symb. Logic* **1**, 103-105.

[RSUV89] R. Ramakrishnan, Y. Sagiv, J. D. Ullman, M. Y. Vardi, 1989. Proof-tree transformation theorems and their applications. *Proc. 8th Symp. on the Principles of Database Systems*, 1989, 172-181.

[Sagi87] Y. Sagiv, 1987. Optimizing Datalog programs. In *Foundations of Deductive Databases and Logic Programming*, ed. Jack Minker, Morgan Kaufmann Publishers, Los Altos, California, 659-698.

[SaYa80] Y. Sagiv, Yannakakis, M., 1980. Equivalence among relational expressions with the union and difference operators. *J. ACM* **27**, 633-655.

[Sar89a] Y. Saraiya, 1989. Linearizing non-linear recursions in polynomial time. *Proc. 8th Symp. on the Principles of Database Systems*, 1989, 182-189.

[Sar89b] Y. Saraiya, 1989. NAIL! seminar, Stanford University, Summer 1989.

[Speh75] J. C. Spehner, 1975. Quelques constructions et algorithmes relatifs aux sous-monoides d'un monoide libre, *Semigroup Forum* **9**, 334-353.

[Thue12] A. Thue, 1914. Probleme über Veränderungen von Zeichenreihen nach gegebenen Regeln, *Skr. Vid. Kristiania, I Mat. Nautruv. Klasse*, No. 10, 34 pages.

[Ullm88] J. D. Ullman, 1988. *Principles of Database and Knowledge-Base Systems, Volume I*, Computer Science Press, Rockville, Maryland.

[Ullm88] J. D. Ullman, 1989. *Principles of Database and Knowledge-Base Systems, Volume II*, Computer Science Press, Rockville, Maryland.

[Ullm85] J. D. Ullman, 1985. Implementation of logical query languages on databases. *ACM Transactions on Database Systems* **10**, 289-321.

[UlVa85] J. D. Ullman, A. Van Gelder, 1985. Parallel Complexity of Logical Query Programs. Stanford University Department of Computer Science Technical Report No. STAN-CS-85-1089.

[UlVa85] J. D. Ullman, A. Van Gelder, 1985. Parallel Complexity of Logical Query Programs. Stanford University Department of Computer Science Technical Report No. STAN-CS-85-1089.

[Vard88] M. Y. Vardi, 1988. Decidability and Undecidability results for boundedness of linear recursive queries. *Proc. 7th Symp. on the Principles of Database Systems*, 1988, 341-351.

[Vard90] M. Y. Vardi, 1990. NAIL! seminar, Stanford University.